**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

================================================================

# A1: Array ADT and its Applications

**Date of submission: 22-09-2021**

**Question:**

1. Create an ADT for an array data structure with the following functions:

a. insertAt(A[ ], size, pos, data) that inserts data at position pos in the array A[size] and returns size of the array if successful or -1 if not successful.

b. search(A[ ], pos, key) that searches key in A[size] starting from pos and return the index of key if found or 0 if not found

c. size(A[ ]) that returns the length of the array a

2. Store arrayADT operations in Array.h
3. Use Array.h and write an application (main.c) for the following:

a. Create a user interface that inserts a set of integers in array ADT. Do not take size of the array as input.

b. Implement insertafterdata(a[ ], data1, data2) that inserts data2 after every occurrence of data1 in a.

c. Write a function printArray(a[ ]) that prints the integers in a with its position horizontally

**Code:**

## array.h

```
//array ADT structure
#define MAX_SIZE 15
#define DELIM -999

int size (int A[]) {
	int count;
	for (count=0; A[count]!=DELIM; count++);
	return count;
	}

int insertAt (int A[], int size, int pos, int data) {
	if (pos>=0 && pos<=size) {
		A[size+1]=DELIM;
		for (int i=size; i>pos; i--)
			A[i]=A[i-1];
		A[pos]=data;
```

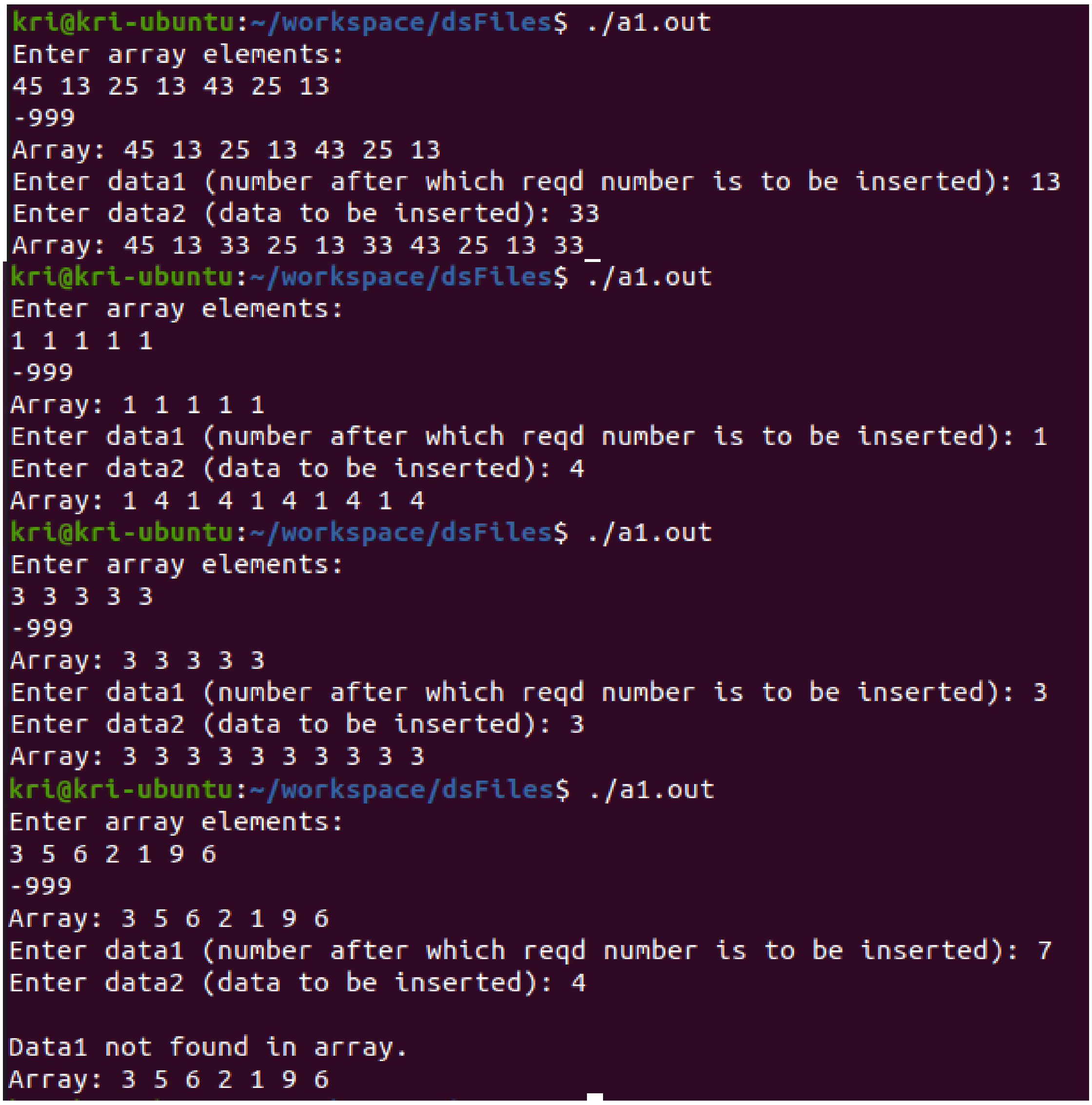
```
        return size+1;
        }
    }

int search (int A[], int pos, int key) {
    if (pos>=0 && pos<size(A)) {
        for (int i=pos; A[i]!=DELIM; i++) {
            if (A[i]==key)
                return i;
            }
        return -1;
        }
    else if (pos<0)
        return -2;
    else if (pos>=size(A))
        return -3;
    }
```

**array.c**

```
//implementation of array ADT
#include <stdio.h>
#include <stdlib.h>
#include "array.h"

void insertafterdata (int a[], int data1, int data2) {
    if (data2!=DELIM) {
        int length=size(a);
        int foundAt=0;
        do {
            foundAt=search(a,foundAt,data1);
            //printf("Found at: %d\n",foundAt);
            if (foundAt!=-1) {
            length=insertAt(a,length,foundAt+1,data2);
            //printf("New size: %d\n",length);
            foundAt+=2;
            }
            } while (foundAt!=length && foundAt!=-1);
        }
    /*else
        printf("Do not enter the delimiter into the array!\n"); //remove*/
    }

void printArray(int a[]) {
    printf("Array: ");
    for (int i=0; i<size(a); i++) {
        printf("%d ",a[i]);
        }
    printf("\n");
    }

int main () {
    //part a
    int data1, data2, A[MAX_SIZE], i=-1, pos;

    printf("Enter array elements:\n");
    do {
        i++;
        scanf("%d",&A[i]);
        } while(A[i]!=DELIM);

    //to check size of array
    //printf("Size of array: %d\n",size(A));

    //to print original array
    printArray(A);

    //part b
    printf("Enter data1 (number after which reqd number is to be inserted):
");
    scanf("%d",&data1);
    printf("Enter data2 (data to be inserted): ");
    scanf("%d",&data2);

    //validation
        //search function
    /*for (i=1; i<=4; i++) {
        printf("Testing search. Enter position to search from: ");
        scanf("%d",&pos);
        int ret=search(A,pos,data1);
        if (ret==-1)
            printf("Data not found in array.\n");
        else if (ret==-2)
            printf("Cannot search as position entered is negative.
Position must be positive and within the array bounds.\n");
        else if (ret==-3)
            printf("Cannot search as position entered is greater than the
size of the array. Position must be positive and within the array bounds.\n");
        else
            printf("Found at index: %d\n",ret);
        }
        //insert function
    for (i=1; i<=3; i++) {
        printf("Testing insert. Enter position to insert at: ");
        scanf("%d",&pos);
        int ret=insertAt(A,size(A),pos,data2);
        if (ret==-1)
            printf("Insert operation was unsuccessful.\n");
        else if (ret==-2)
            printf("Cannot insert as position entered is negative. Position
must be positive and within the array bounds.\n");
        else if (ret==-3)
            printf("Cannot insert as position entered is greater than the
size of the array. Position must be positive and within the array bounds.\n");
```

```
		else
			printf("Size after inserting: %d\n",ret);
		}*/


	//function call to insert data in reqd positions of array
	insertafterdata(A,data1,data2);

	int ret=search(A,0,data1);	//validation
	if (ret==-1)
		printf("\nData1 not found in array.\n");

	//part c
	//function to print new array
	printArray(A);

	return 0;
	}
```

**Output:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ ./a1.out
Enter array elements:
45 13 25 13 43 25 13
-999
Array: 45 13 25 13 43 25 13
Enter data1 (number after which reqd number is to be inserted): 13
Enter data2 (data to be inserted): 33
Array: 45 13 33 25 13 33 43 25 13 33
kri@kri-ubuntu:~/workspace/dsFiles$ ./a1.out
Enter array elements:
1 1 1 1 1
-999
Array: 1 1 1 1 1
Enter data1 (number after which reqd number is to be inserted): 1
Enter data2 (data to be inserted): 4
Array: 1 4 1 4 1 4 1 4 1 4
kri@kri-ubuntu:~/workspace/dsFiles$ ./a1.out
Enter array elements:
3 3 3 3 3
-999
Array: 3 3 3 3 3
Enter data1 (number after which reqd number is to be inserted): 3
Enter data2 (data to be inserted): 3
Array: 3 3 3 3 3 3 3 3 3 3
kri@kri-ubuntu:~/workspace/dsFiles$ ./a1.out
Enter array elements:
3 5 6 2 1 9 6
-999
Array: 3 5 6 2 1 9 6
Enter data1 (number after which reqd number is to be inserted): 7
Enter data2 (data to be inserted): 4

Data1 not found in array.
Array: 3 5 6 2 1 9 6
```

================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

==============================================================

## A2: List ADT using pointers and its Applications

**Date of submission: 29-10-2021**

**Question:**

1. Create an ADT for a List data structure using pointers, with the following functions (List.h):

a. Create a linked list of names (Max. size of the string is 4)

b. Create the following operations

| | |
|---|---|
| insertLast (linklist, item) | to insert the item in the end |
| insertFirst (linklist, item) | to insert the item in the front |
| deleteMiddle(linklist, item) | to delete the first occurrence of the item |
| deleteFirst (linklist) | to delete the first node and return the item |
| deleteLast (linklast) | to delete the last node and return the item |
| insertMiddle(linklist, item1,item2) | to insert item2 after the last occurrence of item1 |
| search (linklist, item) | to return TRUE/FALSE if found/not found |
| length (linklist) | to return the length of the list |
| getData (linklist) | to return name in the node pointed by linklist |

2. Use List.h and write an application (a2List.c) for the following:

a. Write an application program in C that includes options (12 options) to read items of lists, to test each of the list operations and the application operation.

b. Implement reverse(linklist) to return the reverse of the items in the given linklist without creating a new linked list

c. Implement createSorted (linklist, item) that inserts an item in the list such that all the items in the list are in lexically ascending order

d. Write a function display(linklist) to display the items in the list

e. Write a function rotateLeft(linklist) that rotates the items to the left

**Code:**

### list.h

```
//list ADT structure

#define EMPTY "none"
#define SIZE 5

struct node {
        char data[SIZE];
        struct node* next;
        };
```

```
void insertFirst (struct node**, char[]);
void insertLast (struct node**, char[]);
void insertMiddle (struct node**, char[], char[]);
char* deleteFirst (struct node**, char[]);
char* deleteLast (struct node**, char[]);
void deleteMiddle (struct node**, char[]);
int search (struct node**, char[]);
int length (struct node**);
char* getData (struct node**);

void insertFirst (struct node* *linklist, char item[]) {
	//enter data into new node
	struct node* newNode = (struct node*) malloc(sizeof(struct node));
	if (!newNode)
		return; //memory error. validate?
	strcpy(newNode->data,item);
	//add new node to head of linked list
	newNode->next = *linklist;
	*linklist = newNode;
	}

void insertLast (struct node* *linklist, char item[]) {
	//enter data into new node
	struct node* newNode = (struct node*) malloc(sizeof(struct node));
	if (!newNode)
		return; //memory error
	strcpy(newNode->data,item);
	newNode->next = NULL;
	//if empty list, make new node the head
	if (!(*linklist)) {
		*linklist = newNode;
		return;
		}
	//find last node and add new node to end of linked list
	struct node* temp = *linklist;
	while (temp->next) {
		temp = temp->next;
		}
	temp->next = newNode;
	}

void insertMiddle (struct node* *linklist, char item1[], char item2[]) {
	//enter data into new node
	struct node* newNode = (struct node*) malloc(sizeof(struct node));
	if (!newNode)
		return; //memory error
	strcpy(newNode->data,item2);

	//find position node
	struct node *curr = *linklist;
		//first occurrence of item1
```

```
    /*while (curr!=NULL && strcmp(curr->data,item1)) {
        curr = curr->next;
        } */

        //last occurrence of item1
    struct node *last = NULL;
    while (curr!=NULL) {
        if (strcmp(curr->data,item1)==0)
            last = curr;
        curr = curr->next;
        }

    //insert new node to linked list
    if (last!=NULL) {
        newNode->next = last->next;
        last->next = newNode;
        }
    }

char* deleteFirst (struct node* *linklist, char item[]) {
    struct node *temp = *linklist;
    //if empty list
    if (!(*linklist))
        return EMPTY;
    //if single element list, delete that node
    if (temp->next == NULL) {
        strcpy(item,temp->data);
        *linklist = NULL;
        free(temp);
        return item;
        }
    //store data of first node temporarily
    strcpy(item,temp->data);
    //shift head to second node and delete first node
    *linklist = (*linklist)->next;
    free(temp);
    //return deleted data
    return item;
    }

char* deleteLast (struct node* *linklist, char item[]) {
    struct node *temp = *linklist, *prev = NULL;
    //if empty list
    if (!(*linklist))
        return EMPTY;
    //if single element list, delete that node
    if (temp->next == NULL) {
        strcpy(item,temp->data);
        *linklist = NULL;
        free(temp);
        return item;
        }
    //find last node and second-last node
    while (temp->next!=NULL) {
        prev = temp;
        temp = temp->next;
        }
    //store data of last node temporarily
    strcpy(item,temp->data);
    //delete last node
    prev->next = NULL;
    free(temp);
    //return deleted data
    return item;
    }

void deleteMiddle (struct node* *linklist, char item[]) {
    struct node *temp = *linklist, *prev;
    //if head node itself contains item to be deleted
    if (temp!=NULL && !strcmp(temp->data,item)) {
        *linklist = temp->next;
        free(temp);
        return;
        }
    //find node that contains item
    while (temp!=NULL && strcmp(temp->data,item)) {
        prev = temp;
        temp = temp->next;
        }
    //delete the node
    if (temp!=NULL) {
        prev->next = temp->next;
        free(temp);
        }
    }

int search (struct node* *linklist, char item[]) {
    struct node* temp = *linklist;
    while (temp!=NULL) {
        if (!strcmp(temp->data,item))
            return 1;
        temp = temp->next;
        }
    return 0;
    }

int length (struct node* *linklist) {
    int size = 0;
    struct node* curr = *linklist;
    while (curr!=NULL) {
        size++;
        curr = curr->next;
        }
    return size;
```

```
        }

char* getData (struct node* *linklist) {
        return (*linklist)->data;
        }
```

**list.c**

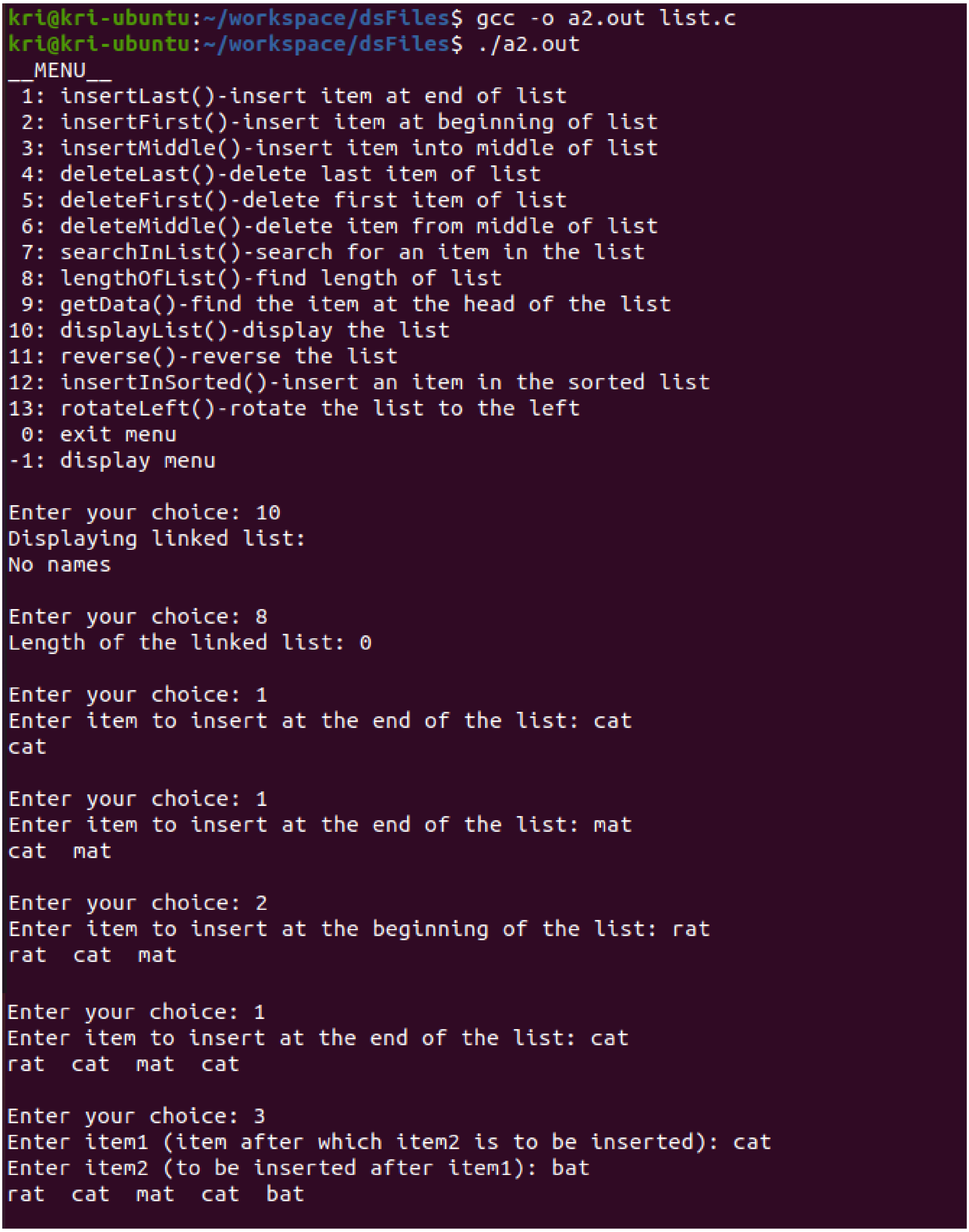
```
//implementation of linked list ADT

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "list.h"

void insertFirst (struct node**, char[]);
void insertLast (struct node**, char[]);
void insertMiddle (struct node**, char[], char[]);
char* deleteFirst (struct node**, char[]);
char* deleteLast (struct node**, char[]);
void deleteMiddle (struct node**, char[]);
int search (struct node**, char[]);
int length (struct node**);
char* getData (struct node**);

void reverse (struct node**);
void createSorted (struct node**, char[]);
void display (struct node*);
void rotateleft (struct node*);
void dispMenu();

void reverse (struct node* *linklist) {
        struct node *prevNode = NULL, *currNode = *linklist, *nextNode =
NULL;
        while (currNode!=NULL) {
                //store next
                nextNode = currNode->next;
                //reverse current node's pointer
                currNode->next = prevNode;
                //move pointers one position ahead
                prevNode = currNode;
                currNode = nextNode;
                }
        *linklist = prevNode;
        }

void createSorted (struct node* *linklist, char item[]) {
        struct node *curr = *linklist, *prev = NULL;
        //if linked list does not exist, add item to list
        if (!(*linklist))
                insertFirst(linklist,item);
        //to find position node to insert new node
```

```
    while (curr!=NULL && (strcmp(curr->data,item)<0)) {
        prev = curr;
        curr = curr->next;
        }
    //to insert item into sorted list at the correct position
    if (prev==NULL)
        insertFirst(linklist,item);
    else if (curr==NULL)
        insertLast(linklist,item);
    else
        insertMiddle(linklist,prev->data,item);
    }

void rotateLeft (struct node* *linklist) {
    char item[SIZE];
    strcpy(item,deleteFirst(linklist,item));
    insertLast(linklist,item);
    }

void display(struct node* linklist) {
    //printf("List: ");
    struct node* temp = linklist;
    if (!linklist) {
        printf("No names\n");
        return;
        }
    while(temp!=NULL) {
        printf("%s  ",temp->data);
        temp = temp->next;
        }
    printf("\n");
    }

int main() {
    //declaring linked list
    struct node* head = (struct node*) malloc(sizeof(struct node));
    head = NULL;

    //implementation of functions - menu form
    int choice;
    dispMenu();
    do {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice){
            case 1: {
                char item[SIZE];
                printf("Enter item to insert at the end of the list: ");
                scanf("%s", item);
                insertLast(&head,item);
                display(head);
                break;
```

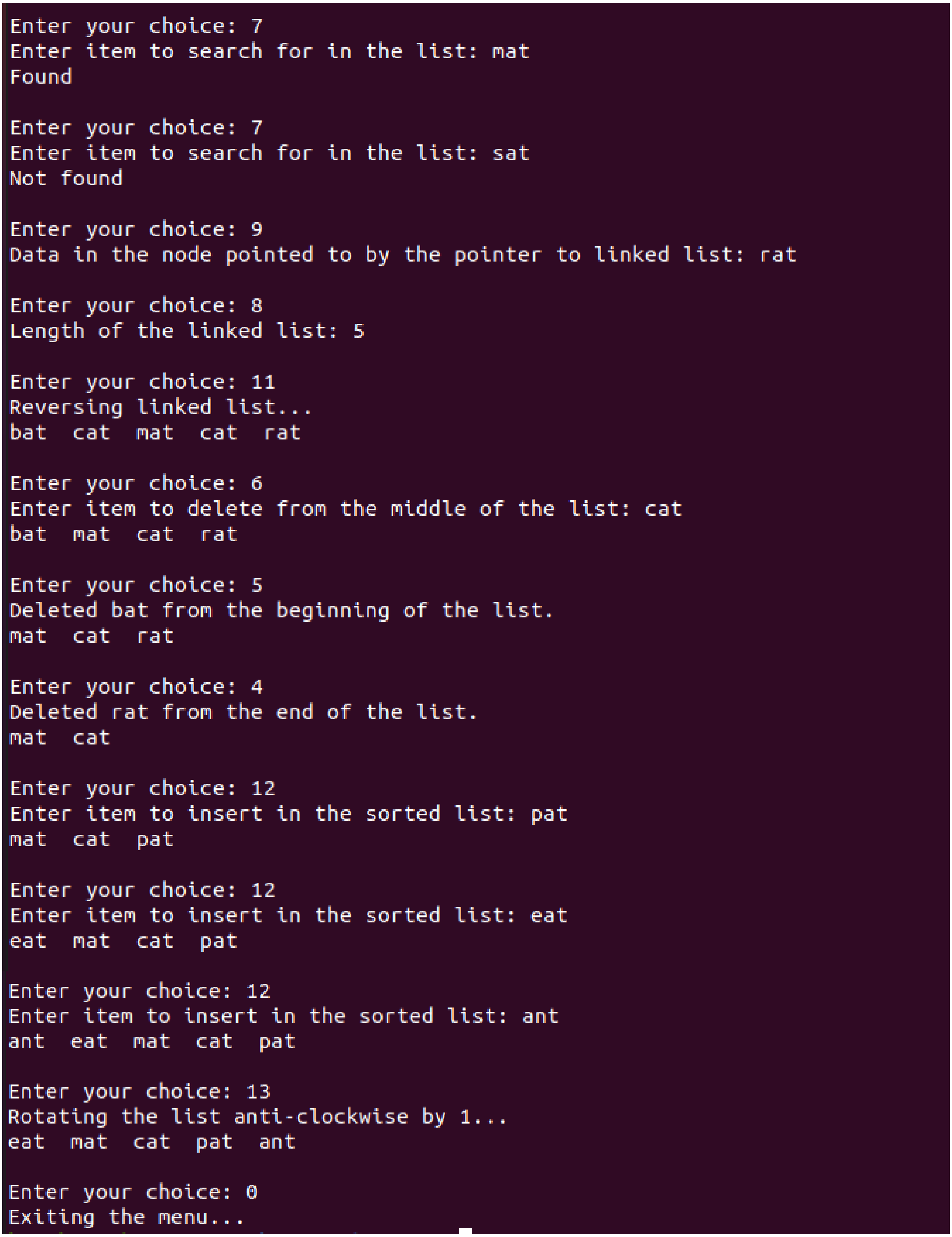
```
                                }

                        case 2: {
                                char item[SIZE];
                                printf("Enter item to insert at the beginning of the list: 
");
                                scanf("%s", item);
                                insertFirst(&head,item);
                                display(head);
                                break;
                                }

                        case 3: {
                                char item1[SIZE];
                                printf("Enter item1 (item after which item2 is to be 
inserted): ");
                                scanf("%s", item1);
                                char item2[SIZE];
                                printf("Enter item2 (to be inserted after item1): ");
                                scanf("%s", item2);
                                insertMiddle(&head, item1, item2);
                                display(head);
                                break;
                                }

                        case 4: {
                                char item[SIZE];
                                strcpy(item,deleteLast(&head,item));
                                if (strcmp(item,EMPTY)==0)
                                        printf("Invalid operation!\n");
                                else
                                        printf("Deleted %s from the end of the list.\n", 
item);
                                display(head);
                                break;
                                }

                        case 5: {
                                char item[SIZE];
                                strcpy(item,deleteFirst(&head,item));
                                if (strcmp(item,EMPTY)==0)
                                        printf("Invalid operation!\n");
                                else
                                        printf("Deleted %s from the beginning of the list.\
n", item);
                                display(head);
                                break;
                                }

                        case 6: {
                                char item[SIZE];
```

```
                printf("Enter item to delete from the middle of the list:
");
                scanf("%s", item);
                deleteMiddle(&head,item);
                display(head);
                break;
                }

            case 7: {
                char item[SIZE];
                printf("Enter item to search for in the list: ");
                scanf("%s", item);
                if (search(&head,item))
                    printf("Found\n");
                else
                    printf("Not found\n");
                break;
                }

            case 8: printf("Length of the linked list: %d\n",
length(&head));
                break;

            case 9: printf("Data in the node pointed to by the pointer to
linked list: %s\n", getData(&head));
                break;

            case 10: {
                printf("Displaying linked list:\n");
                display(head);
                break;
                }

            case 11: {
                printf("Reversing linked list...\n");
                reverse(&head);
                display(head);
                break;
                }

            case 12: {
                char item[SIZE];
                printf("Enter item to insert in the sorted list: ");
                scanf("%s", item);
                createSorted(&head,item);
                display(head);
                break;
                }

            case 13: {
                printf("Rotating the list anti-clockwise by 1...\n");
                rotateLeft(&head);
                display(head);
                break;
                }

            case 0: printf("Exiting the menu...\n");
                break;

            case -1: dispMenu();
                break;

            default: printf("Invalid option. Choice must be an integer in
the range [-1,13].\n"); break;
            }

        } while (choice != 0);
    return 0;
    }

void dispMenu() {
    printf("__MENU__\n");
    printf(" 1: insertLast()-insert item at end of list\n 2: insertFirst()-insert
item at beginning of list\n");
    printf(" 3: insertMiddle()-insert item into middle of list\n 4: deleteLast()-
delete last item of list\n");
    printf(" 5: deleteFirst()-delete first item of list\n 6: deleteMiddle()-delete
item from middle of list\n");
    printf(" 7: searchInList()-search for an item in the list\n 8: lengthOfList()-
find length of list\n");
    printf(" 9: getData()-find the item at the head of the list \n10:
displayList()-display the list\n");
    printf("11: reverse()-reverse the list\n12: insertInSorted()-insert an item
in the sorted list\n");
    printf("13: rotateLeft()-rotate the list to the left\n 0: exit menu\n-1:
display menu\n");
    }
```

**Output:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a2.out list.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a2.out
__MENU__
 1: insertLast()-insert item at end of list
 2: insertFirst()-insert item at beginning of list
 3: insertMiddle()-insert item into middle of list
 4: deleteLast()-delete last item of list
 5: deleteFirst()-delete first item of list
 6: deleteMiddle()-delete item from middle of list
 7: searchInList()-search for an item in the list
 8: lengthOfList()-find length of list
 9: getData()-find the item at the head of the list
10: displayList()-display the list
11: reverse()-reverse the list
12: insertInSorted()-insert an item in the sorted list
13: rotateLeft()-rotate the list to the left
 0: exit menu
-1: display menu

Enter your choice: 10
Displaying linked list:
No names

Enter your choice: 8
Length of the linked list: 0

Enter your choice: 1
Enter item to insert at the end of the list: cat
cat

Enter your choice: 1
Enter item to insert at the end of the list: mat
cat  mat

Enter your choice: 2
Enter item to insert at the beginning of the list: rat
rat  cat  mat

Enter your choice: 1
Enter item to insert at the end of the list: cat
rat  cat  mat  cat

Enter your choice: 3
Enter item1 (item after which item2 is to be inserted): cat
Enter item2 (to be inserted after item1): bat
rat  cat  mat  cat  bat
```



```
Enter your choice: 7
Enter item to search for in the list: mat
Found

Enter your choice: 7
Enter item to search for in the list: sat
Not found

Enter your choice: 9
Data in the node pointed to by the pointer to linked list: rat

Enter your choice: 8
Length of the linked list: 5

Enter your choice: 11
Reversing linked list...
bat  cat  mat  cat  rat

Enter your choice: 6
Enter item to delete from the middle of the list: cat
bat  mat  cat  rat

Enter your choice: 5
Deleted bat from the beginning of the list.
mat  cat  rat

Enter your choice: 4
Deleted rat from the end of the list.
mat  cat

Enter your choice: 12
Enter item to insert in the sorted list: pat
mat  cat  pat

Enter your choice: 12
Enter item to insert in the sorted list: eat
eat  mat  cat  pat

Enter your choice: 12
Enter item to insert in the sorted list: ant
ant  eat  mat  cat  pat

Enter your choice: 13
Rotating the list anti-clockwise by 1...
eat  mat  cat  pat  ant

Enter your choice: 0
Exiting the menu...
```

===============================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

=================================================================

## A3: Applications of Types of Linked Lists – CLL, DLL, CDLL

**Date of submission: 03-11-2021**

**Question:**

1. Create an ADT for a Circular linked list (CLL) of integers where CLL is a pointer to the last node. (CO1, K3)

a) Add the following operations:

insertFirst(CLL, item), insertLast(CLL, item), deleteFirst(CLL), deleteLast(CLL)

b) Write an application for the following that uses CLL

I. rotate(CLL,direction, count) that rotates the integers in the list by left or right (direction is -1 for left; +1 for right) by count number of positions.

II. display(CLL) that displays the integers in the CLL

c) Demonstrate the CLL operations and applications with suitable test cases

2. Create an ADT for a doubly linked list (DLL) of integers where DLL is a pointer to the first node of the DLL. (CO1, K3)

a) Add the following operations:

insertFirst(DLL, item), insertLast(DLL, item), deleteFirst(DLL), deleteLast(DLL)

b) Write an application for the following that uses DLL

I. display(DLL) that displays the integers in the DLL

II. shift(DLL, direction, count) that rotates the integers in the list by left or right (direction is -1 for left; +1 for right) by count number of positions.

III. Demonstrate the DLL operations and applications with suitable test cases

3. The Josephus Problem (CO1,K5)
A group of soldiers are surrounded by an overwhelming enemy force. There is no hope for victory without reinforcement, but there is a single horse available for escape. The soldiers agree to a pact to determine which one of them is to escape and summon help. They stand in a circle and each one chooses a positive integer. One of their names and a positive integer ‘n’ are chosen. Starting with the person whose name is chosen; they count around the circle clockwise and eliminate the nth person. The positive integer which that person chose is then used to continue the count, but this time in the anticlockwise direction. Each time that a person is eliminated, the number the person chosen is used to determine the next person to be eliminated and the direction of traversal is opposite to that of the previous one. i.e. the counting alternates between clockwise and anticlockwise direction.
For example, suppose that the 5 soldiers are A, B, C, D, and E. They chose integers 4,5,6,7, and 8 respectively. The name C and integer 2 are initially chosen. Then the order in which the soldiers are eliminated from the circle is D, A, B, and E leaving C as the last one and C will be signaled to escape.

What to deliver?
1. Write an algorithm for the above problem and implement the algorithm in C. Decide a suitable data structure. Trace the algorithm diagrammatically.
2. Write the ADT for your data structure with suitable operations in a header file.
3. Implement the application separately, making use of the ADT.
4. Write more test cases and test your program.
5. Design a user-friendly interface.

**Code:**

**1)**

**cll.h**

```
//Circular linked list ADT structure

#define EMPTY 0

typedef struct node {
        int data;
        struct node* next;
        } cll_node;

typedef struct node* cll_ptr;

void insertFirst (cll_ptr*,int);
void insertLast (cll_ptr*,int);
int deleteFirst (cll_ptr*);
int deleteLast (cll_ptr*);

void insertFirst (cll_ptr* CLL, int item) {
        //enter data into new node
        cll_ptr newNode = (cll_ptr) malloc(sizeof(cll_node));
        if (!newNode)
                return; //memory error
        newNode->data = item;
        //if empty list, make new node the first/last node (CLL)
        if (*CLL == NULL) {
                *CLL = newNode;
                (*CLL)->next = *CLL;
                return;
                }
        //add new node to front of CLL
        newNode->next = (*CLL)->next;
        (*CLL)->next = newNode;
        }

void insertLast (cll_ptr* CLL, int item) {
        //enter data into new node
        cll_ptr newNode = (cll_ptr) malloc(sizeof(cll_node));
        if (!newNode)
                return; //memory error
        newNode->data = item;
        //if empty list, make new node the first/last node (CLL)
        if (*CLL == NULL) {
                *CLL = newNode;
                (*CLL)->next = *CLL;
                return;
                }
        //add new node to end of CLL
        newNode->next = (*CLL)->next;
        (*CLL)->next = newNode;
        *CLL = newNode;
        }

int deleteFirst (cll_ptr* CLL) {
        //if empty list
        if (!(*CLL))
                return EMPTY;
        //store data of first node temporarily
        cll_ptr temp = (*CLL)->next;
        int item = temp->data;
        //if single element list, delete that node
        if (temp == *CLL) {
                *CLL = NULL;
                free(temp);
                return item;
                }
        //connect last node to second node and delete first node
        (*CLL)->next = temp->next;
        free(temp);
        //return deleted data
        return item;
        }

int deleteLast (cll_ptr* CLL) {
        //if empty list
        if (!(*CLL))
                return EMPTY;
        //store data of last node temporarily
        cll_ptr last = *CLL, temp = *CLL;
        int item = last->data;
        //if single element list, delete that node
        if ((*CLL)->next == *CLL) {
                *CLL = NULL;
                free(last);
                return item;
                }
        //traverse list to find second-last node
        while (temp->next != *CLL) {
                temp = temp->next;
                }
        //connect second-last node to first node and delete last node
        temp->next = (*CLL)->next;
        *CLL = temp;
        free(last);
        //return deleted data
        return item;
        }
```

**cll.c**

```
//implementation of circular linked list ADT

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

int rotate (cll_ptr*,int,int);
void display (cll_ptr);
void dispMenu();

int rotate (cll_ptr* CLL, int direction, int count) {
        if (direction!=1 && direction!=-1)
                return -1;
        if (count < 0)
                return -2;
        int skip = 0, item;
        if (direction == -1)
                while (skip < count) {
                        item = deleteFirst(CLL);
                        insertLast(CLL,item);
                        skip++;
                        }
        else if (direction == 1)
                while (skip < count) {
                        item = deleteLast(CLL);
                        insertFirst(CLL,item);
                        skip++;
                        }
        return 0;
        }

void display (cll_ptr CLL) {
        //printf("List: ");
        if (!CLL) {
                printf("No names\n");
                return;
                }
        cll_ptr temp = CLL->next;
        do {
                printf("%d  ",temp->data);
                temp = temp->next;
                } while (temp != CLL->next);
        printf("\n");
        }
```

```
int main() {
        //declaring CLL
        cll_ptr l1 = NULL;

        //implementation of functions - menu form
        int choice;
        dispMenu();
        do {
                printf("\nEnter your choice: ");
                scanf("%d", &choice);
                switch (choice){
                        case 1: {
                                int item;
                                printf("Enter integer item to insert at the end of the
list: ");
                                scanf("%d",&item);
                                insertLast(&l1,item);
                                display(l1);
                                break;
                                }

                        case 2: {
                                int item;
                                printf("Enter integer item to insert at the beginning of
the list: ");
                                scanf("%d",&item);
                                insertFirst(&l1,item);
                                display(l1);
                                break;
                                }

                        case 3: {
                                int item = deleteLast(&l1);
                                if (item == EMPTY)
                                        printf("Invalid operation! List is empty.\n");
                                else
                                        printf("Deleted %d from the end of the list.\n",
item);
                                display(l1);
                                break;
                                }

                        case 4: {
                                int item = deleteFirst(&l1);
                                if (item == EMPTY)
                                        printf("Invalid operation! List is empty.\n");
                                else
                                        printf("Deleted %d from the beginning of the
list.\n", item);
                                display(l1);
                                break;
                }

            case 5: {
                int dirn, count, result;
                printf("Enter direction in which to rotate list (-1 for left(AC), 1 for right(C)): ");
                scanf("%d",&dirn);
                printf("Enter no. of positions to rotate by (a positive integer): ");
                scanf("%d",&count);
                result = rotate(&l1,dirn,count);
                if (result == 0)
                    printf("Operation successfully completed. List rotated as specified.\n");
                else if (result == -1)
                    printf("Invalid input! Direction must be entered as 1 or -1.\n");
                else if (result == -2)
                    printf("Invalid input! No. of positions to rotate by must be a positive integer.\n");
                display(l1);
                break;
                }

            case 6: {
                printf("Displaying circular linked list:\n");
                display(l1);
                break;
                }

            case 0: printf("Exiting the menu...\n");
                break;

            case -1: dispMenu();
                break;

            default: printf("Invalid option. Choice must be an integer in the range [-1,6].\n"); break;
            }
        } while (choice != 0);

    return 0;
    }

void dispMenu() {
    printf("__MENU__\n");
    printf(" 1: insertLast()-insert item at end of list\n 2: insertFirst()-insert item at beginning of list\n");
    printf(" 3: deleteLast()-delete last item of list\n 4: deleteFirst()-delete first item of list\n");
    printf(" 5: rotateList()-rotate the list\n 6: displayList()-display the list\n");
    printf(" 0: exit menu\n-1: display menu\n");
```

```
	}
```

**2)**

**dll.h**

```
//Doubly linked list ADT structure

#define EMPTY 0

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

typedef struct node* dll_ptr;

void insertFirst (dll_ptr*,int);
void insertLast (dll_ptr*,int);
int deleteFirst (dll_ptr*);
int deleteLast (dll_ptr*);

void insertFirst (dll_ptr* DLL, int item) {
	//enter data into new node
	dll_ptr newNode = (dll_ptr) malloc(sizeof(dll_node));
	if (!newNode)
		return; //memory error
	newNode->data = item;
	//if empty list, make new node the first/last node (DLL)
	if (!(*DLL)) {
		*DLL = newNode;
		(*DLL)->right = NULL;
		(*DLL)->left = NULL;
		return;
		}
	//add new node to front of DLL
	newNode->right = *DLL;
	newNode->left = NULL;
	(*DLL)->left = newNode;
	*DLL = newNode;
	}

void insertLast (dll_ptr* DLL, int item) {
	//enter data into new node
	dll_ptr newNode = (dll_ptr) malloc(sizeof(dll_node));
	if (!newNode)
		return; //memory error
	newNode->data = item;
	//if empty list, make new node the first/last node (DLL)
	if (!(*DLL)) {
		*DLL = newNode;
```

```
                (*DLL)->right = NULL;
                (*DLL)->left = NULL;
                return;
                }
        //add new node to end of DLL
        dll_ptr temp = *DLL;
        newNode->right = NULL;
        while (temp->right != NULL)
                temp = temp->right;
        temp->right = newNode;
        newNode->left = temp;
        }

int deleteFirst (dll_ptr* DLL) {
        //if empty list
        if (!(*DLL))
                return EMPTY;
        //store data of first node temporarily
        dll_ptr temp = *DLL;
        int item = temp->data;
        //if single element list, delete that node
        if (temp->right == NULL) {
                *DLL = NULL;
                free(temp);
                return item;
                }
        //delete first node
        *DLL = (*DLL)->right;
        (*DLL)->left = NULL;
        temp->right = NULL;
        free(temp);
        //return deleted data
        return item;
        }

int deleteLast (dll_ptr* DLL) {
        //if empty list
        if (!(*DLL))
                return EMPTY;
        dll_ptr temp = *DLL;
        int item;
        //if single element list, delete that node
        if ((*DLL)->right == NULL) {
                *DLL = NULL;
                item = temp->data;
                free(temp);
                return item;
                }
        //traverse list to find last and second-last nodes
        dll_ptr prev = NULL;
        while (temp->right != NULL) {
                prev = temp;
                temp = temp->right;
                }
        //store data of last node temporarily
        item = temp->data;
        //delete last node
        temp->left = NULL;
        prev->right = NULL;
        free(temp);
        //return deleted data
        return item;
        }
```

**dll.c**

```
//implementation of doubly linked list ADT

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

int shift (dll_ptr*,int,int);
void display (dll_ptr);
void dispMenu();

int shift (dll_ptr* DLL, int direction, int count) {
        if (direction!=1 && direction!=-1)
                return -1;
        if (count < 0)
                return -2;
        int skip = 0, item;
        if (direction == -1)
                while (skip < count) {
                        item = deleteFirst(DLL);
                        insertLast(DLL,item);
                        skip++;
                        }
        else if (direction == 1)
                while (skip < count) {
                        item = deleteLast(DLL);
                        insertFirst(DLL,item);
                        skip++;
                        }
        return 0;
        }

void display (dll_ptr DLL) {
        //printf("List: ");
        if (!DLL) {
                printf("No names\n");
                return;
                }
        dll_ptr temp = DLL;
        do {
                printf("%d  ",temp->data);
                temp = temp->right;
                } while (temp != NULL);
        printf("\n");
        }

int main() {
        //declaring DLL
        dll_ptr l1 = NULL;

        //implementation of functions - menu form
        int choice;
        dispMenu();
        do {
                printf("\nEnter your choice: ");
                scanf("%d", &choice);
                switch (choice){
                        case 1: {
                                int item;
                                printf("Enter integer item to insert at the end of the
list: ");
                                scanf("%d",&item);
                                insertLast(&l1,item);
                                display(l1);
                                break;
                                }

                        case 2: {
                                int item;
                                printf("Enter integer item to insert at the beginning of
the list: ");
                                scanf("%d",&item);
                                insertFirst(&l1,item);
                                display(l1);
                                break;
                                }

                        case 3: {
                                int item = deleteLast(&l1);
                                if (item == EMPTY)
                                        printf("Invalid operation! List is empty.\n");
                                else
                                        printf("Deleted %d from the end of the list.\n",
item);
                                display(l1);
                                break;
                                }

                        case 4: {
                                int item = deleteFirst(&l1);
                                if (item == EMPTY)
```

```
                    printf("Invalid operation! List is empty.\n");
                else
                    printf("Deleted %d from the beginning of the
list.\n", item);
                display(l1);
                break;
                }

            case 5: {
                int dirn, count, result;
                printf("Enter direction in which to rotate list (-1 for
left(AC), 1 for right(C)): ");
                scanf("%d",&dirn);
                printf("Enter no. of positions to rotate by (a positive
integer): ");
                scanf("%d",&count);
                result = shift(&l1,dirn,count);
                if (result == 0)
                    printf("Operation successfully completed. List
rotated as specified.\n");
                else if (result == -1)
                    printf("Invalid input! Direction must be entered
as 1 or -1.\n");
                else if (result == -2)
                    printf("Invalid input! No. of positions to rotate by
must be a positive integer.\n");
                display(l1);
                break;
                }

            case 6: {
                printf("Displaying doubly linked list:\n");
                display(l1);
                break;
                }

            case 0: printf("Exiting the menu...\n");
                break;

            case -1: dispMenu();
                break;

            default: printf("Invalid option. Choice must be an integer in
the range [-1,6].\n"); break;
            }
        } while (choice != 0);

    return 0;
    }

void dispMenu() {
    printf("__MENU__\n");
```

```
        printf(" 1: insertLast()-insert item at end of list\n 2: insertFirst()-insert item at beginning of list\n");
        printf(" 3: deleteLast()-delete last item of list\n 4: deleteFirst()-delete first item of list\n");
        printf(" 5: shiftList()-rotate the list\n 6: displayList()-display the list\n");
        printf(" 0: exit menu\n-1: display menu\n");
        }
```

**3)**

**cdll.h**

```
//Circular doubly linked list ADT structure

#define EMPTY 0

typedef struct node {
        int data;
        char label;
        struct node* right;
        struct node* left;
        } cdll_node;

typedef struct node* cdll_ptr;

void insertLast (cdll_ptr*,int,char);
int deleteNode (cdll_ptr*,char);

void insertLast (cdll_ptr* CDLL, int item, char holder) {
        //enter data into new node
        cdll_ptr newNode = (cdll_ptr) malloc(sizeof(cdll_node));
        if (!newNode)
                return; //memory error
        newNode->data = item;
        newNode->label = holder;
        //if empty list, make new node the first/last node (CDLL)
        if (*CDLL == NULL) {
                *CDLL = newNode;
                (*CDLL)->right = *CDLL;
                (*CDLL)->left = *CDLL;
                return;
                }
        //add new node to end of CDLL
        newNode->right = (*CDLL)->right;
        (*CDLL)->right = newNode;
        newNode->left = *CDLL;
        (newNode->right)->left = newNode;
        *CDLL = newNode;
        }

int deleteNode (cdll_ptr* CDLL, char holder) {
        //if empty list
        if (!(*CDLL))
                return EMPTY;
        cdll_ptr temp = *CDLL, prev = (*CDLL)->left;
        //if single element list and label is holder, delete that node
        if (((*CDLL)->right == *CDLL) && (temp->label == holder)) {
                *CDLL = NULL;
                int item = temp->data;
                free(temp);
                //return deleted data
                return item;
                }
        //traverse list to find node with holder's label
        do {
                prev = prev->right;
                temp = temp->right;
                } while ((temp != *CDLL) && (temp->label != holder));
        //connect prev node to following node and delete current node
        if (temp->label == holder) {
                int item = temp->data;
                if (temp == *CDLL)
                        *CDLL = prev;
                cdll_ptr next = temp->right;
                prev->right = next;
                next->left = prev;
                free(temp);
                //return deleted data
                return item;
                }
        //if holder not found
        return -1;
        }

int size (cdll_ptr CDLL) {
        int count = 0;
        if (!CDLL)
                return count;
        cdll_ptr temp = CDLL;
        do {
                count++;
                temp = temp->right;
                } while (temp != CDLL);
        return count;
        }

cdll_ptr search (cdll_ptr CDLL, char holder) {
        if (!CDLL)
                return NULL;
        cdll_ptr temp = CDLL;
        do {
                if (temp->label == holder)
                        return temp;
                temp = temp->right;
                } while (temp != CDLL);
        if (temp == CDLL)
                return NULL;
        }
```

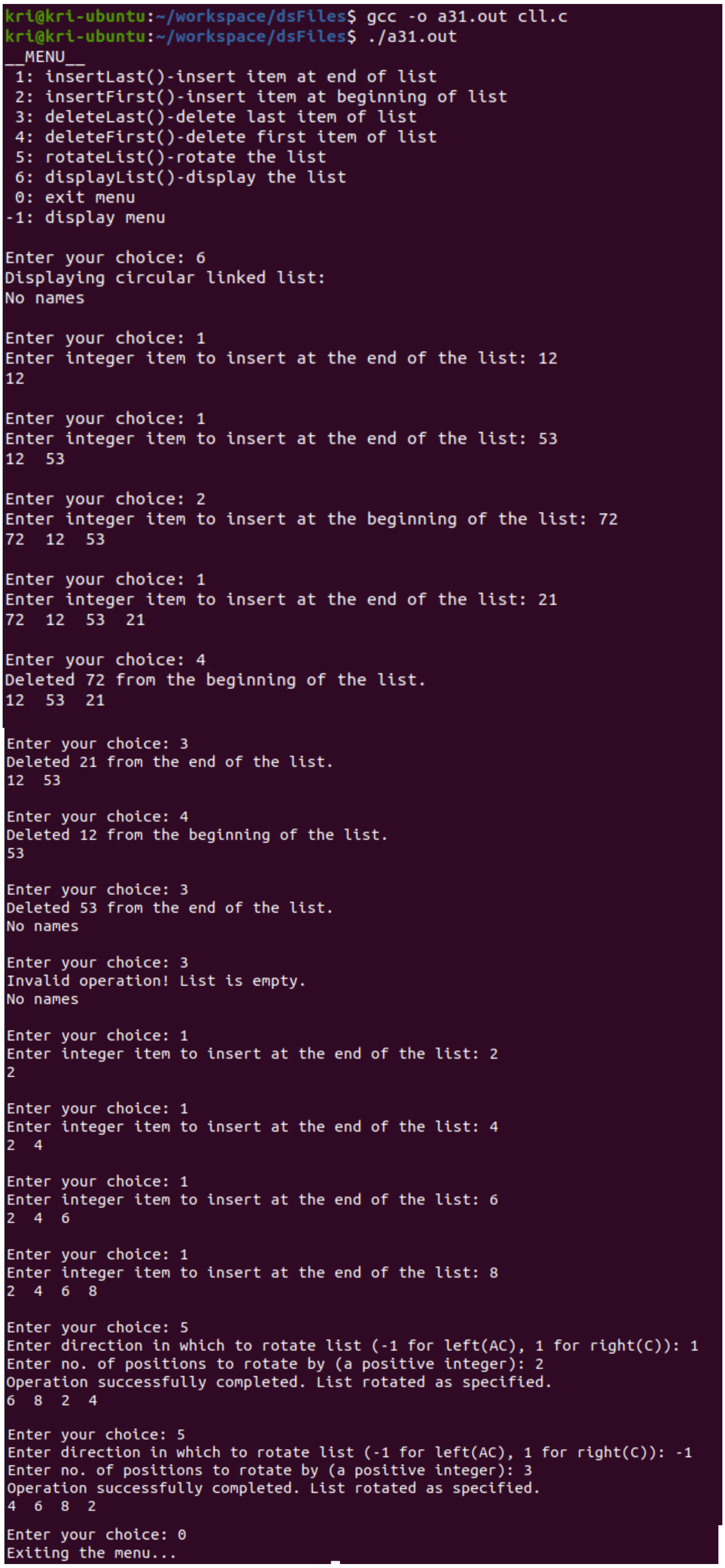
**josephus.c**

```
//josephus problem - implemented using circular doubly linked list ADT

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

void emptyCircle(cdll_ptr*);
void createCircle(cdll_ptr*,int);
char findSafe(cdll_ptr,char,int);
void display(cdll_ptr);
void dispMenu();

void createCircle (cdll_ptr* CDLL, int number) {
        printf("\nEnter every soldier's label and the corresponding number
chosen by them in the right order. Soldiers will be inserted one by one at the
end of the list.\n");
        int num; char name;
        for (int i=0; i<number; i++) {
                printf("\tSoldier%d (A,B,C,...,Z): ",(i+1));
                scanf(" %c",&name);
                while ( (int)name < 65 || (int)name > 90 ) {
                        printf("\tInvalid label! Enter soldier (A,B,C,...,Z): ");
                        scanf(" %c",&name);
                        }
                while (search(*CDLL,name)) {
                        printf("\tSoldier already in circle! Enter unique label
(A,B,C,...,Z): ");
                        scanf(" %c",&name);
                        }
                printf("\tPositive integer chosen: ");
                scanf("%d",&num);
                while (num <= 0) {
                        printf("\tInvalid number! Enter positive integer chosen: ");
                        scanf("%d",&num);
                        }
                insertLast(CDLL,num,name);
                }
        }

void emptyCircle (cdll_ptr* CDLL) {
        while (*CDLL) {
                deleteNode(CDLL,(*CDLL)->label);
                }
        }
```

```
char findSafe (cdll_ptr CDLL, char startName, int startNum) {
	char soldier = startName;
	if (CDLL) {
		cdll_ptr tempCircle = NULL, curr = CDLL->right;
		do {
			insertLast(&tempCircle,curr->data,curr->label);
			curr = curr->right;
			} while (curr != CDLL->right);
		cdll_ptr temp = search(tempCircle,soldier), newStart;
		int sz = size(tempCircle), count = startNum, k;
		for (int i=1; i<sz; i++) { //while (CDLL->right != CDLL)
			if (i%2) {
				for (k=1; k<count; k++)
					temp = temp->right;
				newStart = temp->left;
				}
			else {
				for (k=1; k<count; k++)
					temp = temp->left;
				newStart = temp->right;
				}
			count = deleteNode(&tempCircle,temp->label);
			temp = newStart;
			}
		soldier = temp->label;
		free(tempCircle);
		}
	return soldier;
	}

void display (cdll_ptr CDLL) {
	printf("\n__CIRCLE__\n");
	if (!CDLL) {
		printf("No names\n");
		return;
		}
	cdll_ptr temp = CDLL->right;
	printf("Soldier: ");
	do {
		printf("%c  ",temp->label);
		temp = temp->right;
		} while (temp != CDLL->right);
	printf("\nInteger: ");
	do {
		printf("%d  ",temp->data);
		temp = temp->right;
		} while (temp != CDLL->right);
	printf("\n");
	}

int main() {
	//initialising gameplay
```

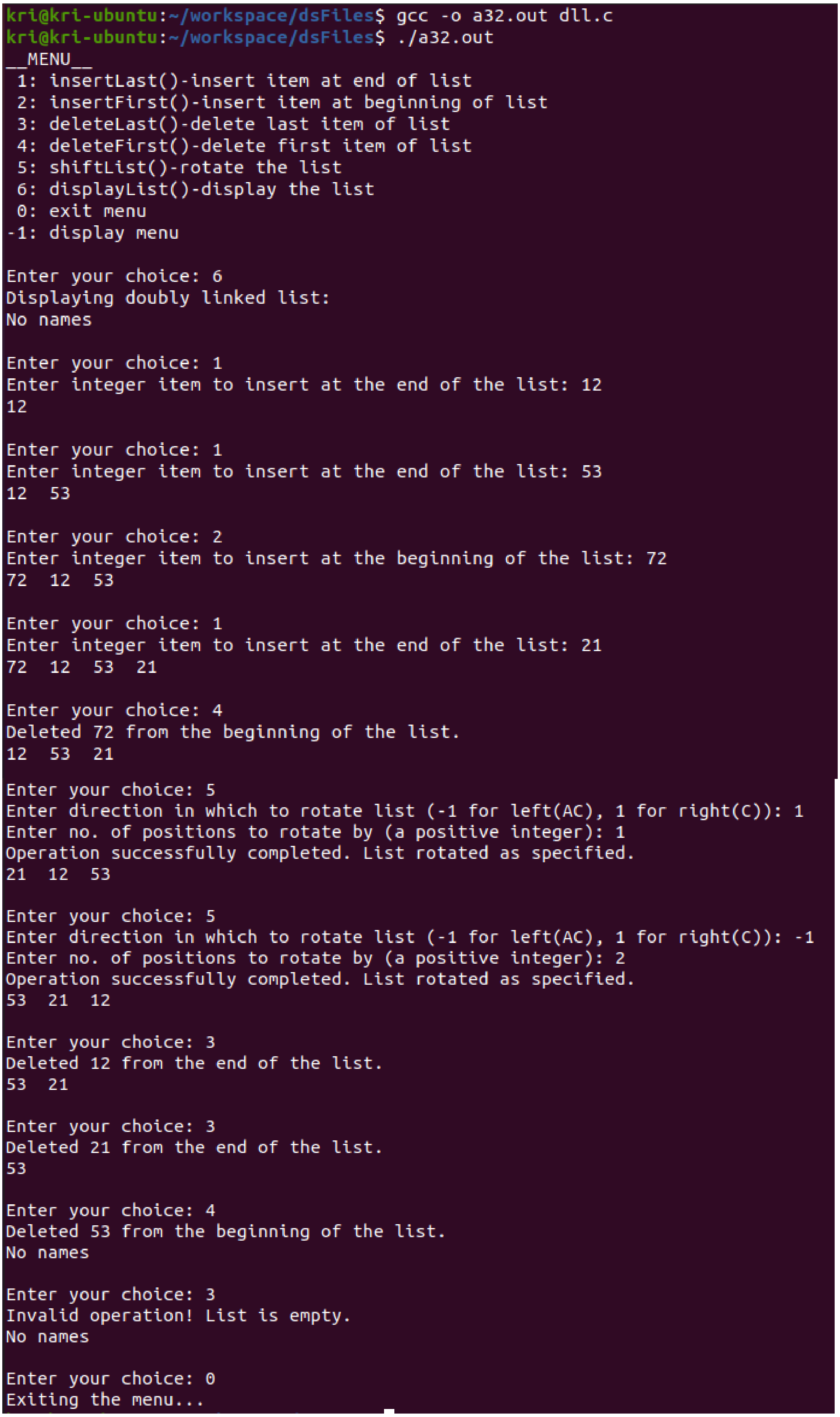
```
    cdll_ptr circle = NULL;

    int choice;
    dispMenu();
    do {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice){
            case 1: {
                int nOfSoldiers;
                circle = NULL;
                printf("\nEnter no. of soldiers in the circle: ");
                scanf("%d",&nOfSoldiers);
                createCircle(&circle,nOfSoldiers);
                display(circle);
                break;
                }

            case 2: {
                emptyCircle(&circle);
                display(circle);
                break;
                }

            case 3: {
                char startName; int startNum;
                printf("Enter the soldier to start with: ");
                scanf(" %c",&startName);
                if (!(search(circle,startName))) {
                    printf("Invalid entry! Soldier not found in circle
or circle does not exist.\n");
                    break;
                    }
                printf("Enter positive integer to start with: ");
                scanf("%d",&startNum);
                if (startNum <= 0) {
                    printf("Invalid entry! Number entered must be a
positive integer.\n");
                    break;
                    }

                //to find the soldier who escapes
                char safe = findSafe(circle,startName,startNum);
                printf("The soldier who escapes is %c.\n",safe);
                break;
                }

            case 4: {
                display(circle);
                break;
                }
            case 0: printf("Exiting the menu...\n");
                break;

            case -1: dispMenu();
                break;

            default: printf("Invalid option. Choice must be an integer in
the range [-1,4].\n"); break;
            }
        } while (choice != 0);

    return 0;
    }

void dispMenu() {
    printf("\n__INTRODUCTION__\n");
    printf("A group of soldiers is surrounded by an overwhelming enemy
force. There is no hope for victory without reinforcement, but there is a single
horse available for escape. The soldiers agree to a pact to determine which
one of them is to escape and summon help. They stand in a circle and each
one chooses a positive integer. One of their names and a positive integer ‘n’
are chosen. Starting with the person whose name is chosen; they count
around the circle clockwise and eliminate the nth person. The positive integer
which that person chose is then used to continue the count, but this time in
the anticlockwise direction. Each time that a person is eliminated, the number
the person chosen is used to determine the next person to be eliminated and
the direction of traversal is opposite to that of the previous one. i.e. the
counting alternates between clockwise and anticlockwise direction.\n");
    printf("\n__INSTRUCTIONS__\n");
    printf("Create different circles of soldiers and start the play with
different soldiers and integers to find the soldier who escapes in each
scenario.\n");
    printf("\n__MENU__\n");
    printf(" 1: createCircle()-create a circle of soldiers\n 2: emptyCicle()-
empty the circle of all soldiers\n");
    printf(" 3: findSafeSoldier()-find the soldier who escapes\n 4:
displayCircle()-display the circle\n");
    printf(" 0: exit the play\n-1: display menu\n");
    }
```

## Output 1:



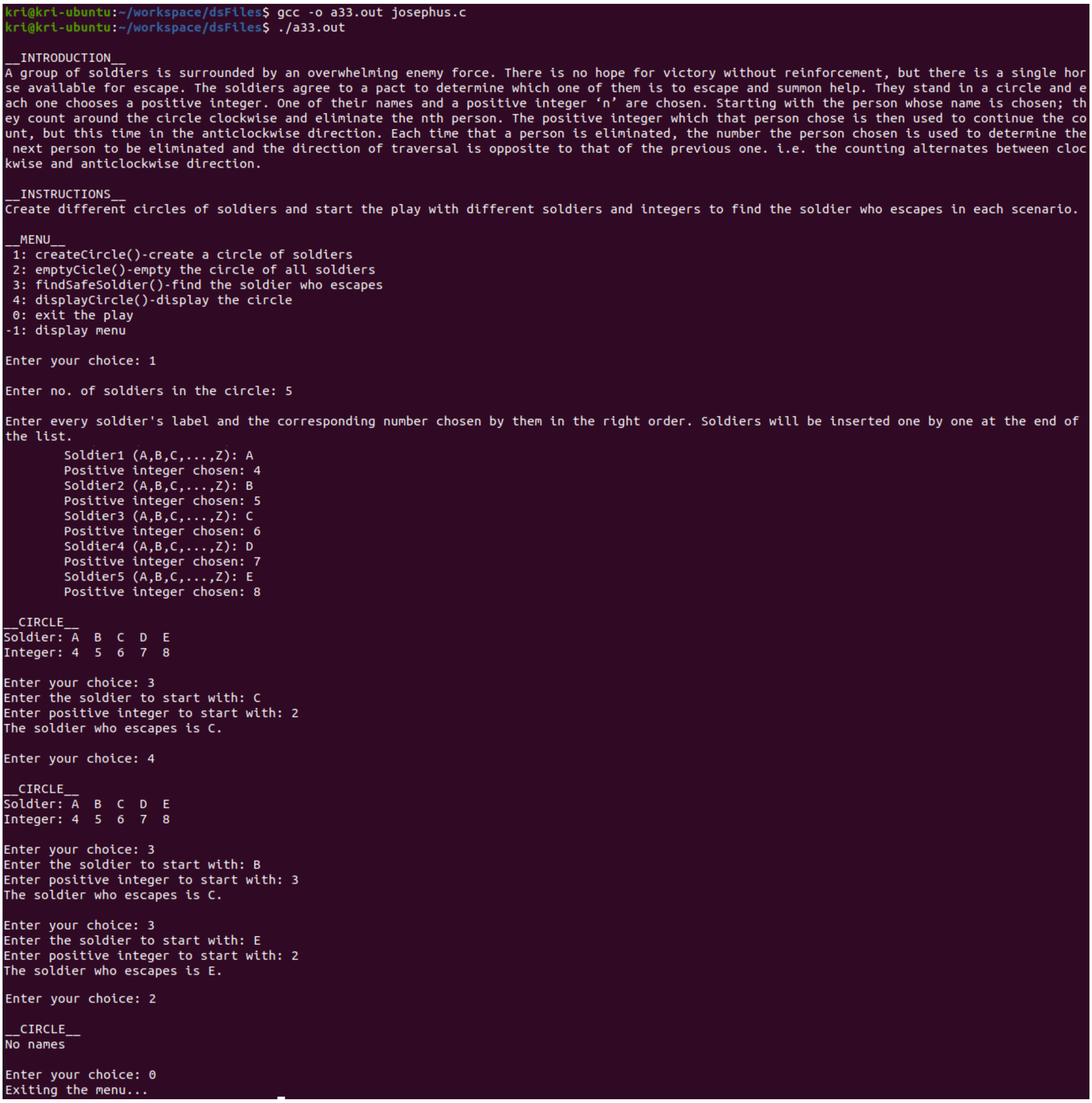
```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a31.out cll.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a31.out
__MENU__
 1: insertLast()-insert item at end of list
 2: insertFirst()-insert item at beginning of list
 3: deleteLast()-delete last item of list
 4: deleteFirst()-delete first item of list
 5: rotateList()-rotate the list
 6: displayList()-display the list
 0: exit menu
-1: display menu

Enter your choice: 6
Displaying circular linked list:
No names

Enter your choice: 1
Enter integer item to insert at the end of the list: 12
12

Enter your choice: 1
Enter integer item to insert at the end of the list: 53
12   53

Enter your choice: 2
Enter integer item to insert at the beginning of the list: 72
72   12   53

Enter your choice: 1
Enter integer item to insert at the end of the list: 21
72   12   53   21

Enter your choice: 4
Deleted 72 from the beginning of the list.
12   53   21

Enter your choice: 3
Deleted 21 from the end of the list.
12   53

Enter your choice: 4
Deleted 12 from the beginning of the list.
53

Enter your choice: 3
Deleted 53 from the end of the list.
No names

Enter your choice: 3
Invalid operation! List is empty.
No names

Enter your choice: 1
Enter integer item to insert at the end of the list: 2
2

Enter your choice: 1
Enter integer item to insert at the end of the list: 4
2   4

Enter your choice: 1
Enter integer item to insert at the end of the list: 6
2   4   6

Enter your choice: 1
Enter integer item to insert at the end of the list: 8
2   4   6   8

Enter your choice: 5
Enter direction in which to rotate list (-1 for left(AC), 1 for right(C)): 1
Enter no. of positions to rotate by (a positive integer): 2
Operation successfully completed. List rotated as specified.
6   8   2   4

Enter your choice: 5
Enter direction in which to rotate list (-1 for left(AC), 1 for right(C)): -1
Enter no. of positions to rotate by (a positive integer): 3
Operation successfully completed. List rotated as specified.
4   6   8   2

Enter your choice: 0
Exiting the menu...
```

**Output 2:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a32.out dll.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a32.out
__MENU__
 1: insertLast()-insert item at end of list
 2: insertFirst()-insert item at beginning of list
 3: deleteLast()-delete last item of list
 4: deleteFirst()-delete first item of list
 5: shiftList()-rotate the list
 6: displayList()-display the list
 0: exit menu
-1: display menu

Enter your choice: 6
Displaying doubly linked list:
No names

Enter your choice: 1
Enter integer item to insert at the end of the list: 12
12

Enter your choice: 1
Enter integer item to insert at the end of the list: 53
12   53

Enter your choice: 2
Enter integer item to insert at the beginning of the list: 72
72   12   53

Enter your choice: 1
Enter integer item to insert at the end of the list: 21
72   12   53   21

Enter your choice: 4
Deleted 72 from the beginning of the list.
12   53   21

Enter your choice: 5
Enter direction in which to rotate list (-1 for left(AC), 1 for right(C)): 1
Enter no. of positions to rotate by (a positive integer): 1
Operation successfully completed. List rotated as specified.
21   12   53

Enter your choice: 5
Enter direction in which to rotate list (-1 for left(AC), 1 for right(C)): -1
Enter no. of positions to rotate by (a positive integer): 2
Operation successfully completed. List rotated as specified.
53   21   12

Enter your choice: 3
Deleted 12 from the end of the list.
53   21

Enter your choice: 3
Deleted 21 from the end of the list.
53

Enter your choice: 4
Deleted 53 from the beginning of the list.
No names

Enter your choice: 3
Invalid operation! List is empty.
No names

Enter your choice: 0
Exiting the menu...
```

## Output 3:



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a33.out josephus.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a33.out

__INTRODUCTION__
A group of soldiers is surrounded by an overwhelming enemy force. There is no hope for victory without reinforcement, but there is a single hor
se available for escape. The soldiers agree to a pact to determine which one of them is to escape and summon help. They stand in a circle and e
ach one chooses a positive integer. One of their names and a positive integer ‘n’ are chosen. Starting with the person whose name is chosen; th
ey count around the circle clockwise and eliminate the nth person. The positive integer which that person chose is then used to continue the co
unt, but this time in the anticlockwise direction. Each time that a person is eliminated, the number the person chosen is used to determine the
 next person to be eliminated and the direction of traversal is opposite to that of the previous one. i.e. the counting alternates between cloc
kwise and anticlockwise direction.

__INSTRUCTIONS__
Create different circles of soldiers and start the play with different soldiers and integers to find the soldier who escapes in each scenario.

__MENU__
 1: createCircle()-create a circle of soldiers
 2: emptyCicle()-empty the circle of all soldiers
 3: findSafeSoldier()-find the soldier who escapes
 4: displayCircle()-display the circle
 0: exit the play
-1: display menu

Enter your choice: 1

Enter no. of soldiers in the circle: 5

Enter every soldier's label and the corresponding number chosen by them in the right order. Soldiers will be inserted one by one at the end of
the list.
        Soldier1 (A,B,C,...,Z): A
        Positive integer chosen: 4
        Soldier2 (A,B,C,...,Z): B
        Positive integer chosen: 5
        Soldier3 (A,B,C,...,Z): C
        Positive integer chosen: 6
        Soldier4 (A,B,C,...,Z): D
        Positive integer chosen: 7
        Soldier5 (A,B,C,...,Z): E
        Positive integer chosen: 8

__CIRCLE__
Soldier: A  B  C  D  E
Integer: 4  5  6  7  8

Enter your choice: 3
Enter the soldier to start with: C
Enter positive integer to start with: 2
The soldier who escapes is C.

Enter your choice: 4

__CIRCLE__
Soldier: A  B  C  D  E
Integer: 4  5  6  7  8

Enter your choice: 3
Enter the soldier to start with: B
Enter positive integer to start with: 3
The soldier who escapes is C.

Enter your choice: 3
Enter the soldier to start with: E
Enter positive integer to start with: 2
The soldier who escapes is E.

Enter your choice: 2

__CIRCLE__
No names

Enter your choice: 0
Exiting the menu...
```

======================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

==================================================================

## A4: Applications of Stacks and Queues

**Date of submission: 03-11-2021**

**Question:**

1. Create an ADT for a Stack of strings (string.h) implemented using arrays. (CO1, K3)
    a) Add the following operations: push, pop, isFull, isEmpty, getTop
    b) Write an application for the following that uses the stack (a4stack.c)
        I. Given an arithmetic expression, convert to postfix and evaluate it.
        II. Given an expression with two types of parenthesis ( [ ] ), check whether the parenthesis are balanced.
    c) Demonstrate the stack operations and applications with suitable test cases
2. Create an ADT for Queue of integers implemented using a circular array (Queue.h) (CO1, K3)
    a) Add the following operations: enqueue. dequeue, isFull, isEmpty, getRear,getFront
    b) Write an application for the following that uses the Queue (a4q.c)
        Consider a Printer Spooler for a network printer — jobs submitted to a printer form a queue for that printer and the job are printed in spooled order. The interface for the printer spooler has the operations: spool, print, list. Implement them using the Queue ADT.
        1. spool(q, jobID) adds a job to q.
        2. print(q) removes the job to be printed from q.
        3. list(q) lists the jobs in q in the spooled order.
    c) Demonstrate the Queue operations and applications with suitable test cases

**Code:**

### stack.h

```
//Stack of characters ADT structure using arrays

#define NUL '\0'

struct Stack {
	int top;
	unsigned capacity;
	char* array;
	};

typedef struct Stack* stack_ptr;

stack_ptr createStack (unsigned);
```

```
int isFull (stack_ptr);
int isEmpty (stack_ptr);
void push (stack_ptr,char);
char pop (stack_ptr);
char getTop (stack_ptr);

stack_ptr createStack (unsigned capacity) {
        stack_ptr stack = (stack_ptr) malloc(sizeof(struct Stack));
        stack->capacity = capacity;
        stack->top = -1;
        stack->array = (char*) malloc(stack->capacity * sizeof(char));
        return stack;
        }

int isFull (stack_ptr stack) {
        //when top equals last index
        return (stack->top == (stack->capacity - 1));
        }

int isEmpty (stack_ptr stack) {
        //when top equals -1
        return (stack->top == -1);
        }

void push (stack_ptr stack, char item) {
        //if stack full
        if (isFull(stack))
                return; //memory error, validate
        //after adding item, top increased by 1
        stack->array[++stack->top] = item;
        }

char pop (stack_ptr stack) {
        if (isEmpty(stack))
                return NUL;
        //returning top item and decreasing top by 1
        return stack->array[stack->top--];
        }

char getTop (stack_ptr stack) {
        if (isEmpty(stack))
                return NUL;
        //returning top item
        return stack->array[stack->top];
        }
```

**stack.c**

```
//implementation of stack of charcters ADT using arrays

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

```
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "stack.h"

#define MAX 30

float toPostfix(char[]);
void evaluate();
int isOperand(char);
int precedence(char);
int checkParantheses(char[]);
int match(char,char);
int checkpt=0;

//function to convert infix expression to postfix
float toPostfix (char exp[]) {
        //declaration of operation stack
        stack_ptr opStack = createStack(strlen(exp));
        stack_ptr exprStack = createStack(strlen(exp));
        if (!opStack || !exprStack)
                return -1;
        if (checkParantheses(exp) != 1)
                return -2;
        if (exp[0]!='(' && exp[0]!='[' && exp[0]!='{')
                return -3;

        for (int i=0; exp[i]; i++) {
                if (isOperand(exp[i]))
                        push(exprStack,exp[i]);
                else if (exp[i] == '(' || exp[i] == '[' || exp[i] == '{')
                        push(opStack,exp[i]);
                else if (exp[i] == ')' || exp[i] == ']' || exp[i] == '}') {
                        while ( !isEmpty(opStack) && getTop(opStack)!='(' )
                                push(exprStack,pop(opStack));
                        pop(opStack);
                        }
                else { //operator
                        char left = exp[i-1], right = exp[i+1];
                        if (!isOperand(left)) {
                                if (left == '(' || left == '[' || left == '{')
                                        return -4;
                                }
                        if (!isOperand(right)) {
                                if (right == ')' || right == ']' || right == '}')
                                        return -4;
                                }
                        while ( !isEmpty(opStack) &&
precedence(exp[i])<=precedence(getTop(opStack)) )
                                push(exprStack,pop(opStack));
                        push(opStack,exp[i]);
                        }
        }

    while ( !isEmpty(opStack) )
        push(exprStack,pop(opStack));

    //exprStack now contains reverse of postFix expression
    stack_ptr inputStack = createStack(strlen(exp));
    while ( !(isEmpty(exprStack)) ) {
        push(inputStack,pop(exprStack));
        }

    //stack for evaluation of postfix
    stack_ptr evalStack = createStack(strlen(exp));

    while ( !(isEmpty(inputStack)) ) {
        char scan = pop(inputStack);
        if (isOperand(scan)) {
            push(evalStack, scan-'0');
            }
        else {
            int val1 = pop(evalStack);
            int val2 = pop(evalStack);
            switch (scan) {
                case '+': push(evalStack, val2 + val1); break;
                case '-': push(evalStack, val2 - val1); break;
                case '*': push(evalStack, val2 * val1); break;
                case '/': push(evalStack, val2/val1); break;
                case '^': push(evalStack, pow(val2,val1)); break;
                }
            }
        }
    float result = pop(evalStack);
    checkpt=1;

    return result;
    }

//function to check if parantheses are balanced or not
int checkParantheses (char exp[]) {
    //declaration of stack
    stack_ptr stack = createStack(strlen(exp));
   char temp;
   for (int i=0; i<strlen(exp); i++) {
       if (exp[i]=='(' || exp[i]=='{' || exp[i]=='[')
           push(stack,exp[i]);
       else if (exp[i]==')' || exp[i]=='}' || exp[i]==']') {
           if (stack->top==-1)   //if stack empty
               return -1;
           else {
               temp=pop(stack);
               if (!match(temp, exp[i]))
                   return -3;
```

```
                    }
                }
        }
       if (stack->top==-1) //if stack empty
             return 1;
       else
             return -2;
        }

int main() {
        //input arithmetic expression
        char exp[MAX];
       printf("Enter an arithmetic expression: ");
       scanf("%s",exp);

        //to convert infix expression to postfix and evaluate

        float result = toPostfix(exp);
        if (result == -1 && checkpt == 0)
                printf("Memory error!\n");
        else if (result == -2 && checkpt == 0) {
                //to check if expression entered is valid or not
                int check = checkParantheses(exp);
                if (check == 1)
                        printf("Parantheses are balanced. The expression is valid.\
n");
                else if (check == -1)
                        printf("The no. of right parentheses is more than the no. of
left parentheses.\nParantheses are not balanced. The expression is invalid.\
n");
                else if (check == -2)
                        printf("The no. of left parentheses is more than the no. of
right parentheses.\nParantheses are not balanced. The expression is invalid.\
n");
                else if (check == -3)
                        printf("Mismatched parentheses.\nParantheses are not
balanced. The expression is invalid.\n");
                else
                        printf("Invalid expression!\n");
                }
        else if (result == -3 && checkpt == 0)
                printf("Expression must be enclosed in parantheses.\n");
        else if (result == -4 && checkpt == 0)
                printf("Operators must be present between operands in an infix
expresion.\n");
        else {
                printf("Result: %f\n",result);
                checkpt=0;
                }

        return 0;
        }
```

```
//function to check if a character is an operand
int isOperand (char ch) {
              return isalnum(ch);
       }

//function to find precedence of an operator
int precedence (char ch) {
              switch (ch) {
                     case '^':
                            { return 3; break; }
                     case '/': case '*': case '%':
                            { return 2; break; }
                     case '+': case '-':
                            { return 1; break; }
                     case '(': case ')': case '[': case ']': case '{': case '}':
                            { return 0; break; }
                     default: return -1;
                     }
       }

//function to check if opening and closing parantheses match
int match (char a,char b) {
      if(a=='[' && b==']')
            return 1;
      if(a=='{' && b=='}')
            return 1;
      if(a=='(' && b==')')
            return 1;
      return 0;
       }
```

**queue.h**

```
//Queue ADT structure using circluar array

struct Queue {
       int front;
       int rear;
       int capacity;
       int *array;
       };

typedef struct Queue* queue_ptr;

queue_ptr createQueue(int);
void enqueue(queue_ptr,int);
int dequeue(queue_ptr);
int isFull(queue_ptr);
int isEmpty(queue_ptr);
int getRear(queue_ptr);
int getFront(queue_ptr);
```

```
queue_ptr createQueue (int size) {
        queue_ptr queue = (queue_ptr) malloc(sizeof(struct Queue));
        queue->capacity = size+1;
        queue->front = -1;
        queue->rear = -1;
        queue->array = (int*) malloc(sizeof(int) * queue->capacity);
        return queue;
        }

void enqueue (queue_ptr queue, int item) {
        int size = queue->capacity;
        /*if (isFull(queue))
                return; //queue full*/
        if (isEmpty(queue)) {
                queue->front = 0;
                queue->rear = 0;
                queue->array[queue->rear++] = item;
                return;
                }
        if (queue->rear == size-1 && queue->front != 0) {
                queue->array[queue->rear] = item;
                queue->rear = 0;
                return;
                }
        queue->array[queue->rear++] = item;
        }

int dequeue (queue_ptr queue) {
        if (isEmpty(queue))
                return -1; //queue empty
        int item = queue->array[queue->front];
        if (queue->front == (queue->rear-1)%(queue->capacity)) {
                queue->front = -1;
                queue->rear = -1;
                }
        else if (queue->front == (queue->capacity)-1)
                queue->front = 0;
        else
                queue->front++;
        return item;
        }

int isFull (queue_ptr queue) {
        int size = queue->capacity;
        return (queue->front == 0 && queue->rear == size-1) || (queue->rear
== (queue->front-1)); //% needed?
        }

int isEmpty (queue_ptr queue) {
        return queue->front == -1;
        }
```

```
int getRear (queue_ptr queue) {
	return queue->array[queue->rear];
	}

int getFront (queue_ptr queue) {
	return queue->array[queue->front];
	}
```

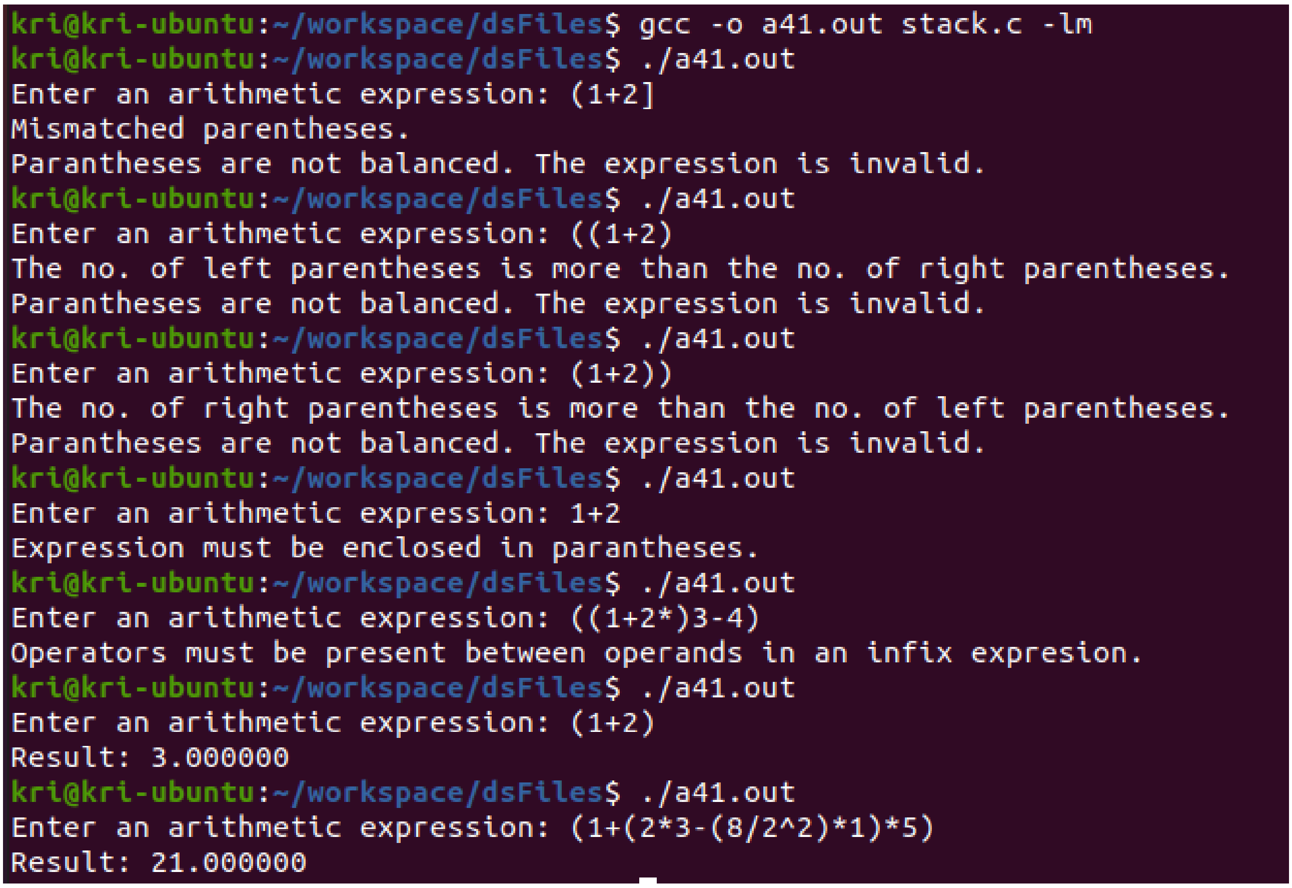
**queue.c**

```
//implementation of queue ADT using circular arrays

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

int spool (queue_ptr,int);
int print (queue_ptr);
void list (queue_ptr);
void dispMenu();

int spool (queue_ptr queue,int jobID) {
	if (isFull(queue))
		return -1; //queue full
	enqueue(queue,jobID);
	return 1;
	}

int print (queue_ptr queue) {
	if (isEmpty(queue))
		return -1; //queue empty
	int jobID = dequeue(queue);
	return jobID;
	}

void list (queue_ptr queue) {
	if (isEmpty(queue)) {
		printf("No jobs in queue.\n");
		return;//validate - printf("Queue is empty.\n");
		}
	//printf("Circular queue: ");
	if (queue->rear >= queue->front) {
		for (int i=queue->front; i<queue->rear; i++)
			printf("%d  ",queue->array[i]);
		}
	else {
		for (int i=queue->front; i<queue->capacity; i++)
			printf("%d  ",queue->array[i]);
		for (int i=0; i<queue->rear; i++)
			printf("%d  ",queue->array[i]);
		}
```

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

int main() {
    //create queue
    printf("__PRINTER SPOOLER__\n");
    int size;
    printf("Enter capacity of printer queue: ");
    scanf("%d",&size);
    queue_ptr queue = createQueue(size);

    //printer spooler interface - menu
    int choice;
    dispMenu();
    do {
        printf("\nEnter choice: ");
        scanf("%d",&choice);

        switch (choice) {
            case 1: {
                int jobID;
                printf("\nEnter job's ID no.: ");
                scanf("%d",&jobID);
                int check = spool(queue,jobID);
                if (check==1)
                    printf("Job submitted successfully.\n");
                else if (check==-1)
                    printf("Queue is full. Wait before addition of a
new job.\n");
                else
                    printf("Error! Operation unsuccessful.\n");
                break;
                }

            case 2: {
                int job = print(queue);
                if (job==-1)
                    printf("Queue is empty. Add some jobs to
perform.\n");
                else
                    printf("Job %d performed successfully.\n",job);
                break;
                }

            case 3: {
                printf("Jobs in queue: ");
                list(queue);
                break;
                }

            case 0: printf("Exiting menu...\n");
                break;
```

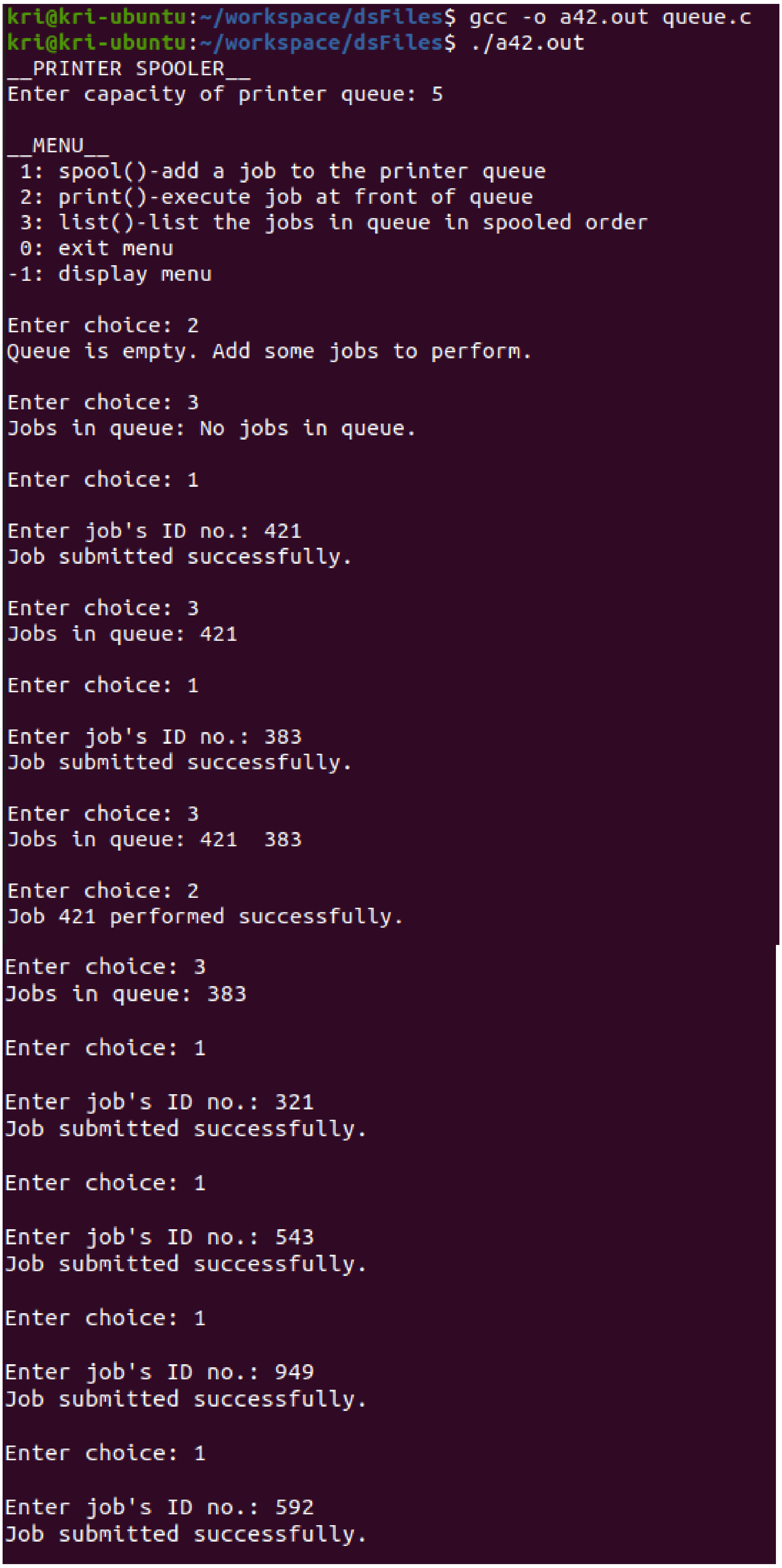
```
                        case -1: dispMenu();
                                break;

                        default: printf("Invalid option! Enter integer in range [-1,3]
only.\n");
                        }
                } while (choice != 0);

        return 0;
        }

void dispMenu() {
        printf("\n__MENU__\n");
        printf(" 1: spool()-add a job to the printer queue\n 2: print()-execute job
at front of queue\n");
        printf(" 3: list()-list the jobs in queue in spooled order\n");
        printf(" 0: exit menu\n-1: display menu\n");
        }
```

**Output 1:**

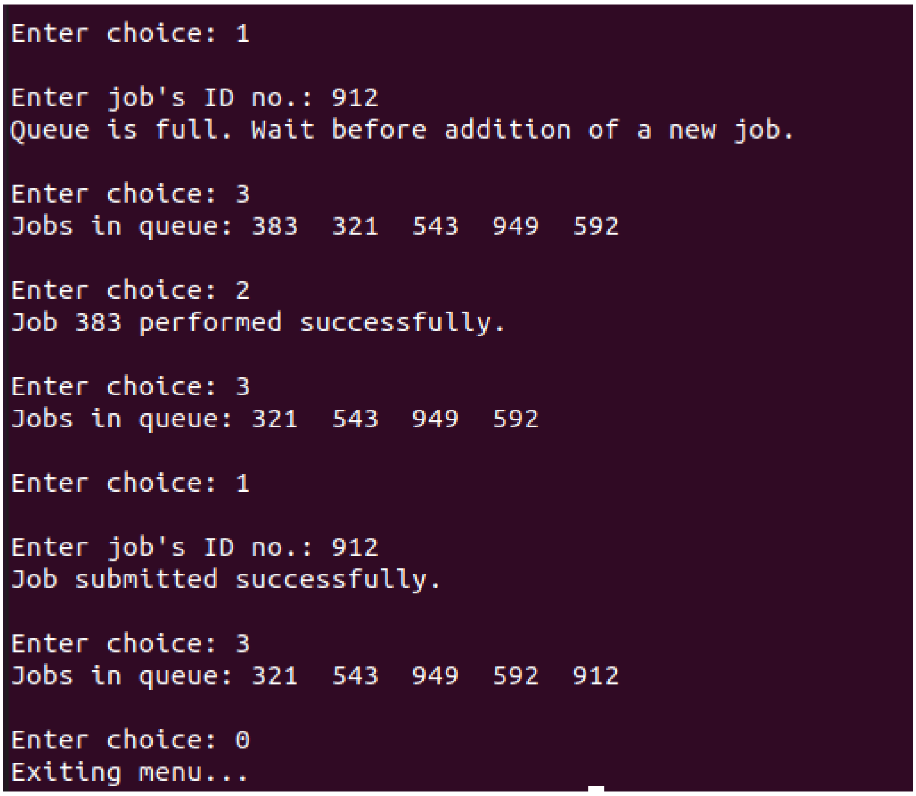


```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a41.out stack.c -lm
kri@kri-ubuntu:~/workspace/dsFiles$ ./a41.out
Enter an arithmetic expression: (1+2]
Mismatched parentheses.
Parantheses are not balanced. The expression is invalid.
kri@kri-ubuntu:~/workspace/dsFiles$ ./a41.out
Enter an arithmetic expression: ((1+2)
The no. of left parentheses is more than the no. of right parentheses.
Parantheses are not balanced. The expression is invalid.
kri@kri-ubuntu:~/workspace/dsFiles$ ./a41.out
Enter an arithmetic expression: (1+2))
The no. of right parentheses is more than the no. of left parentheses.
Parantheses are not balanced. The expression is invalid.
kri@kri-ubuntu:~/workspace/dsFiles$ ./a41.out
Enter an arithmetic expression: 1+2
Expression must be enclosed in parantheses.
kri@kri-ubuntu:~/workspace/dsFiles$ ./a41.out
Enter an arithmetic expression: ((1+2*)3-4)
Operators must be present between operands in an infix expresion.
kri@kri-ubuntu:~/workspace/dsFiles$ ./a41.out
Enter an arithmetic expression: (1+2)
Result: 3.000000
kri@kri-ubuntu:~/workspace/dsFiles$ ./a41.out
Enter an arithmetic expression: (1+(2*3-(8/2^2)*1)*5)
Result: 21.000000
```

**Output 2:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a42.out queue.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a42.out
__PRINTER SPOOLER__
Enter capacity of printer queue: 5

__MENU__
 1: spool()-add a job to the printer queue
 2: print()-execute job at front of queue
 3: list()-list the jobs in queue in spooled order
 0: exit menu
-1: display menu

Enter choice: 2
Queue is empty. Add some jobs to perform.

Enter choice: 3
Jobs in queue: No jobs in queue.

Enter choice: 1

Enter job's ID no.: 421
Job submitted successfully.

Enter choice: 3
Jobs in queue: 421

Enter choice: 1

Enter job's ID no.: 383
Job submitted successfully.

Enter choice: 3
Jobs in queue: 421  383

Enter choice: 2
Job 421 performed successfully.

Enter choice: 3
Jobs in queue: 383

Enter choice: 1

Enter job's ID no.: 321
Job submitted successfully.

Enter choice: 1

Enter job's ID no.: 543
Job submitted successfully.

Enter choice: 1

Enter job's ID no.: 949
Job submitted successfully.

Enter choice: 1

Enter job's ID no.: 592
Job submitted successfully.
```



```
Enter choice: 1

Enter job's ID no.: 912
Queue is full. Wait before addition of a new job.

Enter choice: 3
Jobs in queue: 383  321  543  949  592

Enter choice: 2
Job 383 performed successfully.

Enter choice: 3
Jobs in queue: 321  543  949  592

Enter choice: 1

Enter job's ID no.: 912
Job submitted successfully.

Enter choice: 3
Jobs in queue: 321  543  949  592  912

Enter choice: 0
Exiting menu...
```

==============================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

==============================================================

# A5: Applications of Binary Trees

**Date of submission: 17-11-2021**

**Question:**

1. Create an ADT for a binary tree (BinTree.h) (CO2, K3)
    a) Add the following operations: Construct, inorder, preorder, postorder, levelorder
    b) Write an application for the following that uses the binary tree (a5binTree.c)
        I. Given an arithmetic expression, convert to postfix (use stack.h)
        II. Represent the postfix expression as a binary tree
        III. Evaluate the expression represented by the tree
        IV. Traverse the tree in inorder, preorder, postorder and level order

**Code:**

## stack.h

```
//Stack of characters ADT structure using arrays

#define NUL '\0'

struct Stack {
        int top;
        unsigned capacity;
        char* array;
        };

typedef struct Stack* stack_ptr;

stack_ptr createStack (unsigned);
int isFull (stack_ptr);
int isEmpty (stack_ptr);
void push (stack_ptr,char);
char pop (stack_ptr);
char getTop (stack_ptr);

stack_ptr createStack (unsigned capacity) {
        stack_ptr stack = (stack_ptr) malloc(sizeof(struct Stack));
        stack->capacity = capacity;
        stack->top = -1;
        stack->array = (char*) malloc(stack->capacity * sizeof(char));
        return stack;
        }

int isFull (stack_ptr stack) {
        //when top equals last index
        return (stack->top == (stack->capacity - 1));
        }

int isEmpty (stack_ptr stack) {
        //when top equals -1
        return (stack->top == -1);
        }

void push (stack_ptr stack, char item) {
        //if stack full
        if (isFull(stack))
                return; //memory error, validate
        //after adding item, top increased by 1
        stack->array[++stack->top] = item;
        }

char pop (stack_ptr stack) {
        if (isEmpty(stack))
                return NUL;
        //returning top item and decreasing top by 1
        return stack->array[stack->top--];
        }

char getTop (stack_ptr stack) {
        if (isEmpty(stack))
                return NUL;
        //returning top item
        return stack->array[stack->top];
        }
```

**intStack.h**

```
//Stack of integers ADT structure using arrays

#define EMPTY -1

struct intStack {
        int top;
        unsigned capacity;
        int* array;
        };

typedef struct intStack* IntStack_ptr;

IntStack_ptr createIntStack (unsigned);
int isFullInt (IntStack_ptr);
int isEmptyInt (IntStack_ptr);
void pushInt (IntStack_ptr,int);
int popInt (IntStack_ptr);
int getTopInt (IntStack_ptr);

IntStack_ptr createIntStack (unsigned capacity) {
        IntStack_ptr stack = (IntStack_ptr) malloc(sizeof(struct intStack));
        stack->capacity = capacity;
        stack->top = -1;
        stack->array = (int*) malloc(stack->capacity * sizeof(int));
        return stack;
        }

int isFullInt (IntStack_ptr stack) {
        //when top equals last index
        return (stack->top == (stack->capacity - 1));
        }

int isEmptyInt (IntStack_ptr stack) {
        //when top equals -1
        return (stack->top == -1);
        }

void pushInt (IntStack_ptr stack, int item) {
        //if stack full
        if (isFullInt(stack))
                return; //memory error, validate
        //after adding item, top increased by 1
        stack->array[++stack->top] = item;
        }

int popInt (IntStack_ptr stack) {
        if (isEmptyInt(stack))
                return EMPTY;
        //returning top item and decreasing top by 1
        return stack->array[stack->top--];
        }

int getTopInt (IntStack_ptr stack) {
        if (isEmptyInt(stack))
                return EMPTY;
        //returning top item
        return stack->array[stack->top];
        }
```

**exprn.h**

```
//expression helper operations

int isOperand(char);
int isOperator(char);
int precedence(char);
int checkParantheses(char[]);
int isMatch(char,char);
int validateExprn(char[]);

//function to check if parantheses are balanced or not
int checkParantheses (char exp[]) {
     //declaration of stack
     stack_ptr stack = createStack(strlen(exp));
    char temp;
    for (int i=0; i<strlen(exp); i++) {
        if (exp[i]=='(' || exp[i]=='{' || exp[i]=='[')
            push(stack,exp[i]);
        else if (exp[i]==')' || exp[i]=='}' || exp[i]==']') {
            if (stack->top==-1)   //if stack empty
                return -1;
            else {
                temp=pop(stack);
                if (!isMatch(temp, exp[i]))
                    return -3;
               }
            }
     }
    if (stack->top==-1) //if stack empty
        return 1;
    else
        return -2;
     }

//function to check if a character is an operand
int isOperand (char ch) {
         return isalnum(ch);
     }

//function to check if a character is a valid operator
int isOperator (char ch) {
         return ( ch=='^' || ch=='/' || ch=='*' || ch=='%' || ch=='+' ||
ch=='-' );
     }

//function to find precedence of an operator
int precedence (char ch) {
         switch (ch) {
              case '^':
                   { return 3; break; }
              case '/': case '*': case '%':
                   { return 2; break; }
              case '+': case '-':
                   { return 1; break; }
              default: return -1;
              }
     }
```

```
//function to check if opening and closing parantheses match
int isMatch (char a,char b) {
        if(a=='[' && b==']')
                return 1;
        if(a=='{' && b=='}')
                return 1;
        if(a=='(' && b==')')
                return 1;
        return 0;
         }

//function to validate expression
int validateExprn (char exp[]) {
         if (checkParantheses(exp) != 1)
                  return -2;
         if (exp[0]!='(' && exp[0]!='[' && exp[0]!='{')
                  return -3;
         for (int i=0; exp[i]; i++) {
                  if (isOperator(exp[i])) {
                                      char left = exp[i-1], right = exp[i+1];
                                      if ((left == '(' || left == '[' || left == '{') || (right == ')' ||
right == ']' || right == '}'))
                                               return -4;
                                      }
                  }
         return 1;
         }
```

**BTnode.h**

```
//binary tree node structure

struct tnode {
         char data;
         struct tnode* left;
         struct tnode* right;
         };

struct tnodeInt {
         int data;
         char item;
         struct tnodeInt* left;
         struct tnodeInt* right;
         };

typedef struct tnode* ptnode;
typedef struct tnodeInt* ptnodeInt;

ptnode insertNode(char);
ptnodeInt insertNodeInt(int);

ptnode insertNode (char value) {
         ptnode newNode = (ptnode) malloc(sizeof(struct tnode));
         if (!newNode)
                 return NULL; //memory error
         newNode->data = value;
         newNode->left = NULL;
         newNode->right = NULL;

         return newNode;
         }

ptnodeInt insertNodeInt(int value) {
         ptnodeInt newNode = (ptnodeInt) malloc(sizeof(struct tnodeInt));
         if (!newNode)
                 return NULL; //memory error
         newNode->data = value;
         newNode->item = '$';
         newNode->left = NULL;
         newNode->right = NULL;

         return newNode;
         }

ptnodeInt insertNodeChar(char value) {
         ptnodeInt newNode = (ptnodeInt) malloc(sizeof(struct tnodeInt));
         if (!newNode)
                 return NULL; //memory error
         newNode->data = 0;
         newNode->item = value;
         newNode->left = NULL;
         newNode->right = NULL;

         return newNode;
         }
```

**BTstack.h**

```
//Stack of tree nodes - ADT structure using arrays

struct btStack {
        int top;
        unsigned capacity;
        ptnode *array;
        };

typedef struct btStack* btStack_ptr;

btStack_ptr createBTStack (unsigned);
int isBTFull (btStack_ptr);
int isBTEmpty (btStack_ptr);
void pushBT (btStack_ptr,ptnode);
ptnode popBT (btStack_ptr);
ptnode getTopBT (btStack_ptr);
```

```
btStack_ptr createBTStack (unsigned capacity) {
	btStack_ptr treeStack = (btStack_ptr) malloc(sizeof(struct btStack));
	treeStack->capacity = capacity;
	treeStack->top = -1;
	treeStack->array = (ptnode*) malloc(treeStack->capacity * sizeof(struct tnode));
	return treeStack;
	}

int isBTFull (btStack_ptr treeStack) {
	//when top equals last index
	return (treeStack->top == (treeStack->capacity - 1));
	}

int isBTEmpty (btStack_ptr treeStack) {
	//when top equals -1
	return (treeStack->top == -1);
	}

void pushBT (btStack_ptr treeStack, ptnode item) {
	//if tree stack full
	if (isBTFull(treeStack))
		return; //memory error, validate
	//after adding item, top increased by 1
	treeStack->array[++treeStack->top] = item;
	}

ptnode popBT (btStack_ptr treeStack) {
	if (isBTEmpty(treeStack))
		return NULL;
	//returning top item and decreasing top by 1
	return treeStack->array[treeStack->top--];
	}

ptnode getTopBT (btStack_ptr treeStack) {
	if (isBTEmpty(treeStack))
		return NULL;
	//returning top item
	return treeStack->array[treeStack->top];
	}

#include "BTstackInt.h"
```

**BTstackInt.h**

```
//Stack of tree nodes with integer values accepted- ADT structure using arrays

struct btStackInt {
	int top;
	unsigned capacity;
	ptnodeInt *array;
```

```
        };

typedef struct btStackInt* btStackInt_ptr;

btStackInt_ptr createBTIntStack (unsigned);
int isBTIntFull (btStackInt_ptr);
int isBTIntEmpty (btStackInt_ptr);
void pushBTInt (btStackInt_ptr,ptnodeInt);
ptnodeInt popBTInt (btStackInt_ptr);

btStackInt_ptr createBTIntStack (unsigned capacity) {
        btStackInt_ptr treeStack = (btStackInt_ptr) malloc(sizeof(struct
btStackInt));
        treeStack->capacity = capacity;
        treeStack->top = -1;
        treeStack->array = (ptnodeInt*) malloc(treeStack->capacity *
sizeof(struct tnode));
        return treeStack;
        }

int isBTIntFull (btStackInt_ptr treeStack) {
        //when top equals last index
        return (treeStack->top == (treeStack->capacity - 1));
        }

int isBTIntEmpty (btStackInt_ptr treeStack) {
        //when top equals -1
        return (treeStack->top == -1);
        }

void pushBTInt (btStackInt_ptr treeStack, ptnodeInt item) {
        //if tree stack full
        if (isBTIntFull(treeStack))
                return; //memory error, validate
        //after adding item, top increased by 1
        treeStack->array[++treeStack->top] = item;
        }

ptnodeInt popBTInt (btStackInt_ptr treeStack) {
        if (isBTIntEmpty(treeStack))
                return NULL;
        //returning top item and decreasing top by 1
        return treeStack->array[treeStack->top--];
        }
```

**BTqueue.h**

```
//Queue of tree nodes - ADT structure using circular array

#define SIZE 20

struct btQueue {
	int front;
	int rear;
	int capacity;
	ptnode *array;
	};

typedef struct btQueue* btQueue_ptr;

btQueue_ptr createBTQueue(int);
void enqueueBT(btQueue_ptr,ptnode);
ptnode dequeueBT(btQueue_ptr);
int isFullBT(btQueue_ptr);
int isEmptyBT(btQueue_ptr);

btQueue_ptr createBTQueue (int size) {
	btQueue_ptr treeQueue = (btQueue_ptr) malloc(sizeof(struct btQueue));
	treeQueue->capacity = size+1;
	treeQueue->front = -1;
	treeQueue->rear = -1;
	treeQueue->array = (ptnode*) malloc(sizeof(struct btQueue) *
treeQueue->capacity);
	return treeQueue;
	}

void enqueueBT (btQueue_ptr treeQueue, ptnode item) {
	int size = treeQueue->capacity;
	if (isFullBT(treeQueue))
		return;
	if (isEmptyBT(treeQueue)) {
		treeQueue->front = 0;
		treeQueue->rear = 0;
		treeQueue->array[treeQueue->rear++] = item;
		return;
		}
	if (treeQueue->rear == size-1 && treeQueue->front != 0) {
		treeQueue->array[treeQueue->rear] = item;
		treeQueue->rear = 0;
		return;
		}
	treeQueue->array[treeQueue->rear++] = item;
	}

ptnode dequeueBT (btQueue_ptr treeQueue) {
	if (isEmptyBT(treeQueue))
		return NULL; //treeQueue empty
	ptnode item = treeQueue->array[treeQueue->front];
	if (treeQueue->front == (treeQueue->rear-1)%(treeQueue->capacity)) {
		treeQueue->front = -1;
		treeQueue->rear = -1;
		}
	else if (treeQueue->front == (treeQueue->capacity)-1)
		treeQueue->front = 0;
```

```
        else
                treeQueue->front++;
        return item;
        }

int isFullBT (btQueue_ptr treeQueue) {
        int size = treeQueue->capacity;
        return (treeQueue->front == 0 && treeQueue->rear == size-1) ||
(treeQueue->rear == (treeQueue->front-1));
        }

int isEmptyBT (btQueue_ptr treeQueue) {
        return treeQueue->front == -1;
        }

#include "BTqueueInt.h"
```

**BTqueueInt.h**

```
//Queue of tree nodes - ADT structure using circular array

#define SIZE 20

struct btQueueInt {
        int front;
        int rear;
        int capacity;
        ptnodeInt *array;
        };

typedef struct btQueueInt* btQueueInt_ptr;

btQueueInt_ptr createBTIntQueue(int);
void enqueueIntBT(btQueueInt_ptr,ptnodeInt);
ptnodeInt dequeueIntBT(btQueueInt_ptr);
int isFullBTInt(btQueueInt_ptr);
int isEmptyBTInt(btQueueInt_ptr);

btQueueInt_ptr createBTIntQueue (int size) {
        btQueueInt_ptr treeQueue = (btQueueInt_ptr) malloc(sizeof(struct
btQueueInt));
        treeQueue->capacity = size+1;
        treeQueue->front = -1;
        treeQueue->rear = -1;
        treeQueue->array = (ptnodeInt*) malloc(sizeof(struct btQueue) *
treeQueue->capacity);
        return treeQueue;
        }

void enqueueIntBT (btQueueInt_ptr treeQueue, ptnodeInt item) {
        int size = treeQueue->capacity;
        if (isFullBTInt(treeQueue))
```

```
		return;
	if (isEmptyBTInt(treeQueue)) {
		treeQueue->front = 0;
		treeQueue->rear = 0;
		treeQueue->array[treeQueue->rear++] = item;
		return;
		}
	if (treeQueue->rear == size-1 && treeQueue->front != 0) {
		treeQueue->array[treeQueue->rear] = item;
		treeQueue->rear = 0;
		return;
		}
	treeQueue->array[treeQueue->rear++] = item;
	}

ptnodeInt dequeueIntBT (btQueueInt_ptr treeQueue) {
	if (isEmptyBTInt(treeQueue))
		return NULL; //treeQueue empty
	ptnodeInt item = treeQueue->array[treeQueue->front];
	if (treeQueue->front == (treeQueue->rear-1)%(treeQueue->capacity)) {
		treeQueue->front = -1;
		treeQueue->rear = -1;
		}
	else if (treeQueue->front == (treeQueue->capacity)-1)
		treeQueue->front = 0;
	else
		treeQueue->front++;
	return item;
	}

int isFullBTInt (btQueueInt_ptr treeQueue) {
	int size = treeQueue->capacity;
	return (treeQueue->front == 0 && treeQueue->rear == size-1) ||
(treeQueue->rear == (treeQueue->front-1));
	}

int isEmptyBTInt (btQueueInt_ptr treeQueue) {
	return treeQueue->front == -1;
	}
```

**binTree.h**

```
//Binary tree ADT structure

#include "BTnode.h"
#include "BTstack.h"
#include "BTqueue.h"

ptnode construct(char*);
void inorder(ptnode);
void preorder(ptnode);
void postorder(ptnode);
void levelorder(ptnode);
void visit(char);
int height(ptnode);

ptnode construct (char* postfix) {
    int size = strlen(postfix);
    btStack_ptr exprStack = createBTStack(size);
    for (int i=0; i<size; i++) {
        char c = postfix[i];
        if (isalnum(c)) {
            pushBT(exprStack,insertNode(c));
            }
        else {
            ptnode head = insertNode(c);
            head->right = popBT(exprStack);
            head->left = popBT(exprStack);
            pushBT(exprStack,head);
            }
        }
    return popBT(exprStack);
    }

void inorder (ptnode root) {
    if (root) {
        inorder(root->left);
        visit(root->data);
        inorder(root->right);
        }
    }

void preorder (ptnode root) {
    if (root) {
        visit(root->data);
        preorder(root->left);
        preorder(root->right);
        }
    }

void postorder (ptnode root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        visit(root->data);
        }
    }

void levelorder (ptnode root) {
    btQueue_ptr levelQueue = createBTQueue(SIZE);
    ptnode EOL = insertNode('|');
    ptnode EOB = insertNode(';');
    ptnode blank = insertNode('_');
    int ht = height(root);
    enqueueBT(levelQueue,EOL);
    int level = 1, k=0, eob=0;
    for (int i=0; i<=(ht-level); i++)
        printf("\t");
    while (root) {
        if (root!=blank && root!=EOB) {
            printf("%c\t\t",root->data);
            if (root->left)
                enqueueBT(levelQueue,root->left);
            else
                enqueueBT(levelQueue,blank);
            if (root->right)
                enqueueBT(levelQueue,root->right);
            else
                enqueueBT(levelQueue,blank);
            enqueueBT(levelQueue,EOB);
            root = dequeueBT(levelQueue);
            }

        while (root == blank) {
            root = dequeueBT(levelQueue);
            printf("\t");
            k++;
            }
        while (root == EOB) {
            root = dequeueBT(levelQueue);
            if (root != EOL) {
                printf("  ");
                eob++;
                }
            }
        if (root == EOL) {
            root = dequeueBT(levelQueue);
            if (root != NULL) {
                enqueueBT(levelQueue,EOL);
                level++;
                printf("\n\n");
                for (int i=0; i<=(ht-level); i++)
                    printf("\t");
                for (int i=0; i<k; i++)
                    printf("\t");
                for (int i=0; i<eob; i++)
                    printf("   ");
                }
            }
        }
    printf("\n");
    }

void visit (char value) {
    printf("%c",value);
```

```
	}

int height (ptnode root) {
	if (!root)
		return 0;
	int leftHt = height(root->left);
	int rightHt = height(root->right);

	if (leftHt > rightHt)
		return leftHt+1;
	else
		return rightHt+1;
	}

int heightInt (ptnodeInt root) {
	if (!root)
		return 0;
	int leftHt = heightInt(root->left);
	int rightHt = heightInt(root->right);

	if (leftHt > rightHt)
		return leftHt+1;
	else
		return rightHt+1;
	}
```

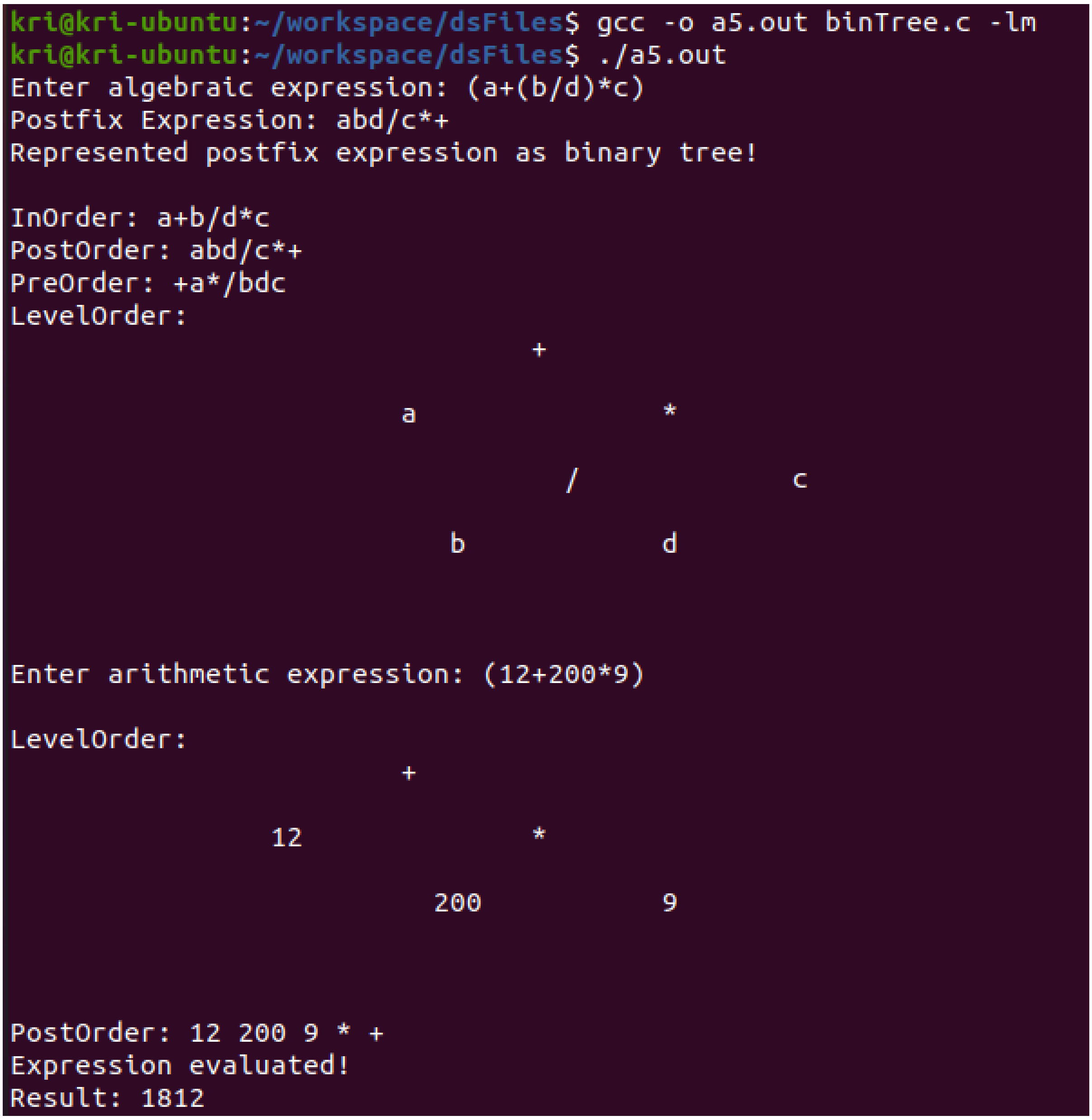
**binTree.c**

```
//implementation of binary tree ADT

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include "stack.h"
#include "intStack.h"
#include "exprn.h"
#include "binTree.h"

char* InfixToPostfix(char[]);
void toIntPostfix (char[],stack_ptr,IntStack_ptr);
ptnodeInt constructIntTree (char[]);
void levelorderInt (ptnodeInt);
void postorderInt(ptnodeInt);
int evalExprn(ptnodeInt);
void validate(int,char[]);

int main() {
	char exp[50], pfExp[50];
	printf("Enter algebraic expression: ");
	scanf("%s", exp);
```

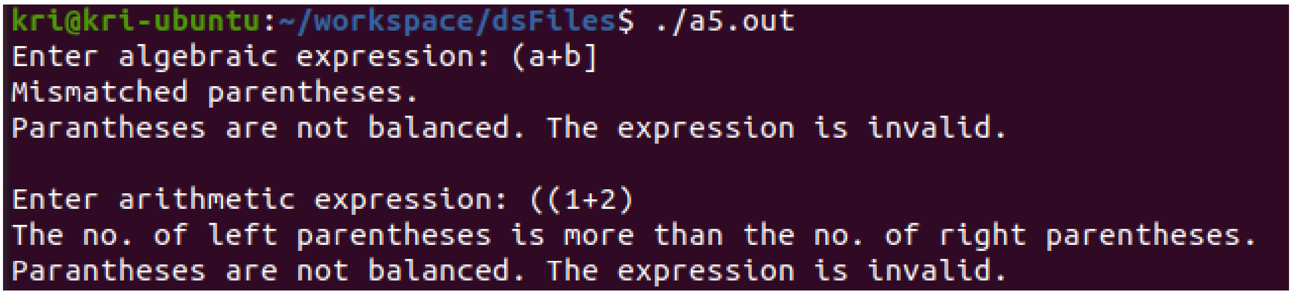
```
    int valid = validateExprn(exp);
    validate(valid,exp);

    if (valid == 1) {
        strcpy(pfExp, InfixToPostfix(exp));
        printf("Postfix Expression: %s\n", pfExp);

        ptnode exprTree = construct(pfExp);
        printf("Represented postfix expression as binary tree!\n");

        printf("\nInOrder: ");
        inorder(exprTree);

        printf("\nPostOrder: ");
        postorder(exprTree);

        printf("\nPreOrder: ");
        preorder(exprTree);

        printf("\nLevelOrder:\n");
        levelorder(exprTree);
        }

    printf("\nEnter arithmetic expression: ");
    scanf("%s",exp);

    valid = validateExprn(exp);
    validate(valid,exp);

    if (valid==1) {
        ptnodeInt tree = constructIntTree(exp);

        printf("\nLevelOrder:\n");
        levelorderInt(tree);

        printf("\nPostOrder: ");
        postorderInt(tree);

        //converting expression to postfix and constructing integer binary tree for evaluation
        int result = evalExprn(tree);
        printf("\nExpression evaluated!\n");
        printf("Result: %d\n",result);
        }

    return 0;
    }

//function to convert infix expression to postfix
char* InfixToPostfix (char exp[]) {
    stack_ptr opStack = createStack(strlen(exp));
    stack_ptr exprStack = createStack(strlen(exp));
    char* postExp = (char*) malloc (sizeof(char) * (strlen(exp)+10));

    if ( !((checkParantheses(exp) != 1) || (exp[0]!='(' && exp[0]!='[' &&
exp[0]!='{')) ) {
        for (int i=0; exp[i]; i++) {
            if (isOperand(exp[i]))
                push(exprStack,exp[i]);
            else if (exp[i] == '(' || exp[i] == '[' || exp[i] == '{')
                push(opStack,exp[i]);
            else if (exp[i] == ')' || exp[i] == ']' || exp[i] == '}') {
                while ( !isEmpty(opStack) && getTop(opStack)!='(' )
                    push(exprStack,pop(opStack));
                pop(opStack);
                }
            else if (isOperator(exp[i])) {
                char left = exp[i-1], right = exp[i+1];
                if ((left == '(' || left == '[' || left == '{') || (right == ')' ||
right == ']' || right == '}'))
                    return (postExp);
                else {
                    while ( !isEmpty(opStack) &&
precedence(exp[i])<=precedence(getTop(opStack)) )
                        push(exprStack,pop(opStack));
                    push(opStack,exp[i]);
                    }
                }
            else
                return (postExp);
            }

        while ( !isEmpty(opStack) )
            push(exprStack,pop(opStack));

        //exprStack now contains reverse of postFix expression
        stack_ptr inputStack = createStack(strlen(exp));
        while ( !(isEmpty(exprStack)) )
            push(inputStack,pop(exprStack));

        int k=-1;
        while ( !(isEmpty(inputStack)) ) {
            postExp[++k] = pop(inputStack);
            }
        postExp[++k] = '\0';
        }
    return (postExp);
    }

//function to convert arithmetic expression to postfix
void toIntPostfix (char exp[], stack_ptr inputStack, IntStack_ptr IntStack) {
    stack_ptr opStack = createStack(strlen(exp));
    stack_ptr expStack = createStack(strlen(exp));
    IntStack_ptr integers = createIntStack(strlen(exp));
    char fill; int skip=0;

    if ( !((checkParantheses(exp) != 1) || (exp[0]!='(' && exp[0]!='[' &&
exp[0]!='{')) ) {
        for (int i=0; exp[i]; i++) {
            if (isOperand(exp[i])) {
                int j, val = exp[i]-'0';
                for (j=i+1; isOperand(exp[j]); j++) {
                    int val2 = exp[j]-'0';
                    val = (val*10)+val2;
                    }
                pushInt(integers,val);
                push(expStack,'a'+skip);
                (++skip)%26;
                i=j-1;
                }
            else if (exp[i] == '(' || exp[i] == '[' || exp[i] == '{')
                push(opStack,exp[i]);
            else if (exp[i] == ')' || exp[i] == ']' || exp[i] == '}') {
                while ( !isEmpty(opStack) && getTop(opStack)!='(' )
                    push(expStack,pop(opStack));
                pop(opStack);
                }
            else if (isOperator(exp[i])) {
                char left = exp[i-1], right = exp[i+1];
                if ((left == '(' || left == '[' || left == '{') || (right == ')' ||
right == ']' || right == '}'))
                    return;
                else {
                    while ( !isEmpty(opStack) &&
precedence(exp[i])<=precedence(getTop(opStack)) )
                        push(expStack,pop(opStack));
                    push(opStack,exp[i]);
                    }
                }
            else
                return;
            }

        while ( !isEmpty(opStack) )
            push(expStack,pop(opStack));

        while ( !(isEmpty(expStack)) )
            push(inputStack,pop(expStack));

        while ( !(isEmptyInt(integers)) )
            pushInt(IntStack,popInt(integers));
        }
    }

//function to add arithmetic expression into binary tree
```

```
ptnodeInt constructIntTree (char exp[]) {
        int size = strlen(exp);
        stack_ptr inputStack = createStack(size);
        IntStack_ptr IntStack = createIntStack(size);
        toIntPostfix(exp,inputStack,IntStack);

        btStackInt_ptr exprStack = createBTIntStack(size);
        while (! isEmpty(inputStack) ) {
                char c = pop(inputStack);
                if (isOperand(c)) {
                        int num = popInt(IntStack);
                        pushBTInt(exprStack,insertNodeInt(num));
                        }
                else {
                        ptnodeInt head = insertNodeChar(c);
                        head->right = popBTInt(exprStack);
                        head->left = popBTInt(exprStack);
                        pushBTInt(exprStack,head);
                        }
                }
        return popBTInt(exprStack);
        }

//function to evaluate postfix expression using integer stack
int evalExprn (ptnodeInt root) {

        if (!root)
                return 0;
        if (!root->left && !root->right)
                return root->data;

        int lValue = evalExprn(root->left);
        int rValue = evalExprn(root->right);

        switch(root->item) {
                case '+': return lValue+rValue;
                case '-': return lValue-rValue;
                case '*': return lValue*rValue;
                case '/': return lValue/rValue;
                case '^': return pow(lValue,rValue);
                case '%': return lValue%rValue;
                case '$': return root->data;
                }
        }

//function to validate input expression
void validate (int result, char exp[]) {
        if (result == -2) {
                //to check if expression entered is valid or not
                int check = checkParantheses(exp);
                if (check == 1)
                printf("Parantheses are balanced. The expression is valid.\
n");
            else if (check == -1)
                printf("The no. of right parentheses is more than the no. of
left parentheses.\nParantheses are not balanced. The expression is invalid.\
n");
            else if (check == -2)
                printf("The no. of left parentheses is more than the no. of
right parentheses.\nParantheses are not balanced. The expression is invalid.\
n");
            else if (check == -3)
                printf("Mismatched parentheses.\nParantheses are not
balanced. The expression is invalid.\n");
            else
                printf("Invalid expression!\n");
            }
        else if (result == -3)
            printf("Expression must be enclosed in parantheses.\n");
        else if (result == -4)
            printf("Operators must be present between operands in an infix
expresion.\n");
        }

//function for postorder traveresal of arithmetic expression tree
void postorderInt (ptnodeInt root) {
        if (root) {
            postorderInt(root->left);
            postorderInt(root->right);
            if ( root->item == '$' )
                printf("%d ",root->data);
            else if (isOperator(root->item))
                printf("%c ",root->item);
            }
        }

//function to print integer expression tree
void levelorderInt (ptnodeInt root) {
        btQueueInt_ptr levelQueue = createBTIntQueue(SIZE);
        ptnodeInt EOL = insertNodeChar('|');
        ptnodeInt EOB = insertNodeChar(';');
        ptnodeInt blank = insertNodeChar('_');
        int ht = heightInt(root);

        enqueueIntBT(levelQueue,EOL);
        int level = 1, k=0, eob=0;
        for (int i=0; i<=(ht-level); i++)
            printf("\t");
        while (root) {
            if (root!=blank && root!=EOB) {
                if ( root->item == '$' )
                    printf("%d\t\t",root->data);
                else if (isOperator(root->item))
            printf("%c\t\t",root->item);
        if (root->left)
            enqueueIntBT(levelQueue,root->left);
        else
            enqueueIntBT(levelQueue,blank);
        if (root->right)
            enqueueIntBT(levelQueue,root->right);
        else
            enqueueIntBT(levelQueue,blank);
        enqueueIntBT(levelQueue,EOB);
        root = dequeueIntBT(levelQueue);
        }

    while (root == blank) {
        root = dequeueIntBT(levelQueue);
        printf("\t");
        k++;
        }
    while (root == EOB) {
        root = dequeueIntBT(levelQueue);
        if (root != EOL) {
            printf("  ");
            eob++;
            }
        }
    if (root == EOL) {
        root = dequeueIntBT(levelQueue);
        if (root != NULL) {
            enqueueIntBT(levelQueue,EOL);
            level++;
            printf("\n\n");
            for (int i=0; i<=(ht-level); i++)
                printf("\t");
            for (int i=0; i<k; i++)
                printf("\t");
            for (int i=0; i<eob; i++)
                printf("   ");
            }
        }
    }
printf("\n");
}
```

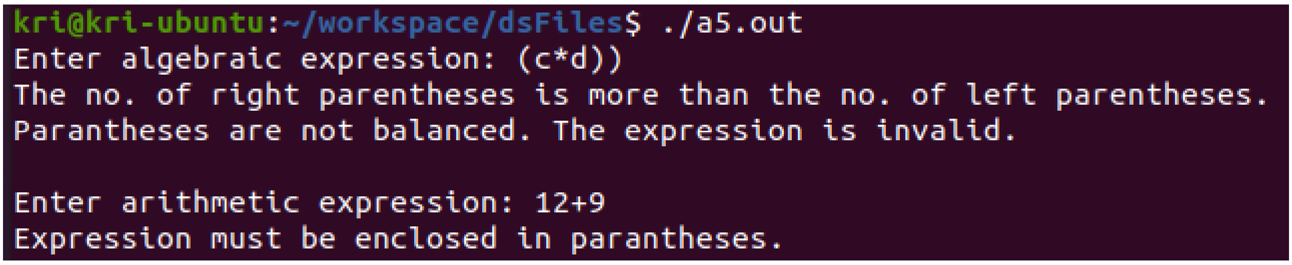
**Output:**



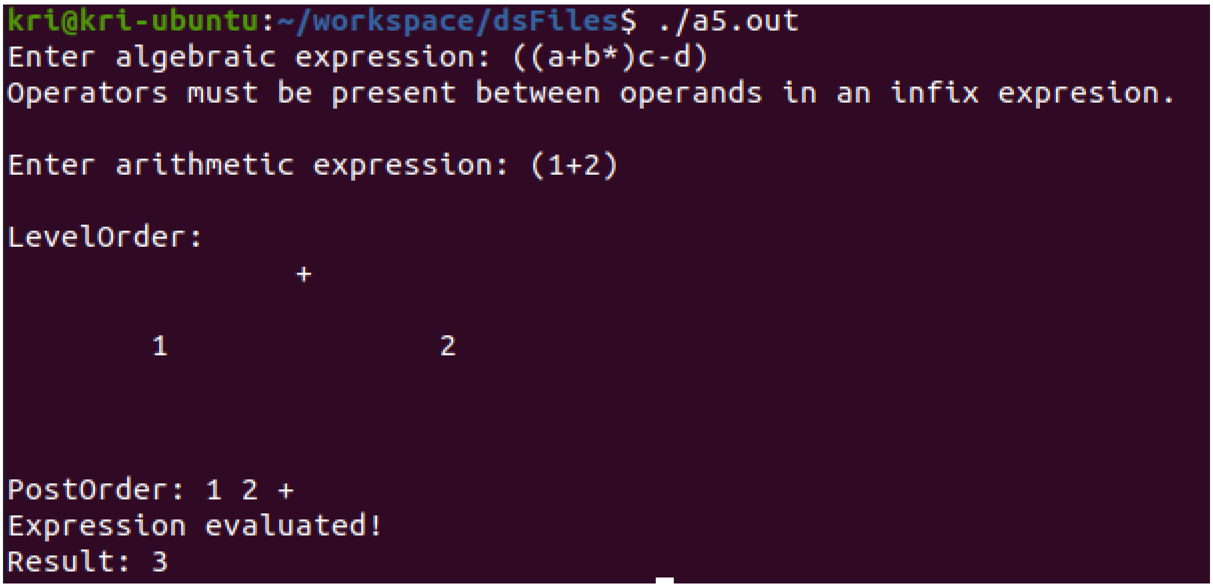
```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a5.out binTree.c -lm
kri@kri-ubuntu:~/workspace/dsFiles$ ./a5.out
Enter algebraic expression: (a+(b/d)*c)
Postfix Expression: abd/c*+
Represented postfix expression as binary tree!

InOrder: a+b/d*c
PostOrder: abd/c*+
PreOrder: +a*/bdc
LevelOrder:
                                +

                        a               *

                                  /             c

                           b            d



Enter arithmetic expression: (12+200*9)

LevelOrder:
                        +

                12              *

                          200           9



PostOrder: 12 200 9 * +
Expression evaluated!
Result: 1812
```



```
kri@kri-ubuntu:~/workspace/dsFiles$ ./a5.out
Enter algebraic expression: (a+b]
Mismatched parentheses.
Parantheses are not balanced. The expression is invalid.

Enter arithmetic expression: ((1+2)
The no. of left parentheses is more than the no. of right parentheses.
Parantheses are not balanced. The expression is invalid.
```



```
kri@kri-ubuntu:~/workspace/dsFiles$ ./a5.out
Enter algebraic expression: (c*d))
The no. of right parentheses is more than the no. of left parentheses.
Parantheses are not balanced. The expression is invalid.

Enter arithmetic expression: 12+9
Expression must be enclosed in parantheses.
```



```
kri@kri-ubuntu:~/workspace/dsFiles$ ./a5.out
Enter algebraic expression: ((a+b*)c-d)
Operators must be present between operands in an infix expresion.

Enter arithmetic expression: (1+2)

LevelOrder:
                +

        1               2



PostOrder: 1 2 +
Expression evaluated!
Result: 3
```

==============================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

====================================================================

## A6: Applications of Binary Search Trees

**Date of submission: 24-11-2021**

**Question:**

1. Create an ADT for a binary search tree (bst.h). (CO2, K3)
    a) Add the following operations:
        Insert, delete, inorder, preorder, postorder, levelorder, search, maximum, minimum
    b) Write an application for the following (a6bst.c)
        I. Check whether two BSTs are identical
        II. Print the number of leaf nodes, non-leaf nodes, total number of nodes
    c) Demonstrate the binary search tree operations and applications with test cases
        Sample Input: 65 34 29 65 10 7 15 1

**Code:**

### bstNode.h

```
//BST node structure

struct bst {
	int key;
	struct bst* left;
	struct bst* right;
	};

typedef struct bst* pnodeBST;

pnodeBST createNode(int);

pnodeBST createNode (int value) {
	pnodeBST newNode = (pnodeBST) malloc(sizeof(struct bst));
	if (!newNode)
		return NULL; //memory error
	newNode->key = value;
	newNode->left = NULL;
	newNode->right = NULL;

	return newNode;
	}
```

**bstQueue.h**

```
//Queue of bst nodes - ADT structure using circular array

#define SIZE 20

struct bstQueue {
        int front;
        int rear;
        int capacity;
        pnodeBST *array;
        };

typedef struct bstQueue* bstQueue_ptr;

bstQueue_ptr createBSTQueue(int);
void enqueueBST(bstQueue_ptr,pnodeBST);
pnodeBST dequeueBST(bstQueue_ptr);
int isFullBST(bstQueue_ptr);
int isEmptyBST(bstQueue_ptr);

bstQueue_ptr createBSTQueue (int size) {
        bstQueue_ptr treeQueue = (bstQueue_ptr) malloc(sizeof(struct
bstQueue));
        treeQueue->capacity = size+1;
        treeQueue->front = -1;
        treeQueue->rear = -1;
        treeQueue->array = (pnodeBST*) malloc(sizeof(struct bstQueue) *
treeQueue->capacity);
        return treeQueue;
        }

void enqueueBST (bstQueue_ptr treeQueue, pnodeBST item) {
        int size = treeQueue->capacity;
        if (isFullBST(treeQueue))
                return;
        if (isEmptyBST(treeQueue)) {
                treeQueue->front = 0;
                treeQueue->rear = 0;
                treeQueue->array[treeQueue->rear++] = item;
                return;
                }
        if (treeQueue->rear == size-1 && treeQueue->front != 0) {
                treeQueue->array[treeQueue->rear] = item;
                treeQueue->rear = 0;
                return;
                }
        treeQueue->array[treeQueue->rear++] = item;
        }

pnodeBST dequeueBST (bstQueue_ptr treeQueue) {
        if (isEmptyBST(treeQueue))
                return NULL; //treeQueue empty
        pnodeBST item = treeQueue->array[treeQueue->front];
        if (treeQueue->front == (treeQueue->rear-1)%(treeQueue->capacity)) {
                treeQueue->front = -1;
                treeQueue->rear = -1;
                }
        else if (treeQueue->front == (treeQueue->capacity)-1)
                treeQueue->front = 0;
        else
                treeQueue->front++;
        return item;
        }

int isFullBST (bstQueue_ptr treeQueue) {
        int size = treeQueue->capacity;
        return (treeQueue->front == 0 && treeQueue->rear == size-1) ||
(treeQueue->rear == (treeQueue->front-1));
        }

int isEmptyBST (bstQueue_ptr treeQueue) {
        return treeQueue->front == -1;
        }
```

**bst.h**

```
//Binary Search Tree ADT structure

#include "bstNode.h"
#include "bstQueue.h"

pnodeBST insertNode(pnodeBST,int);
pnodeBST deleteNode(pnodeBST,int);
void inorder(pnodeBST);
void preorder(pnodeBST);
void postorder(pnodeBST);
void levelorder(pnodeBST);
pnodeBST search(pnodeBST,int);
pnodeBST maximum(pnodeBST);
pnodeBST minimum(pnodeBST);
void visit(int);
int height(pnodeBST);

pnodeBST insertNode (pnodeBST root, int key) {
        if (!root)
                return createNode(key);
        if (key <= root->key)
                root->left = insertNode(root->left,key);
        else if (key > root->key)
                root->right = insertNode(root->right,key);
        return root;
        }
```

```
pnodeBST deleteNode (pnodeBST root, int key) {
        //base case
        if (!root)
                return root;

        //find position of key to be deleted
        if (key < root->key)
                root->left = deleteNode(root->left,key);
        else if (key > root->key)
                root->right = deleteNode(root->right,key);
        //node found
        else {
                //node with one child or no children
                if (!root->left) {
                        pnodeBST temp = root->right;
                        free(root);
                        return temp;
                        }
                else if (!root->right) {
                        pnodeBST temp = root->left;
                        free(root);
                        return temp;
                        }
                //node with two children
                pnodeBST temp = minimum(root->right); //replace root with
smallest in right subtree
                root->key = temp->key;
                root->right = deleteNode(root->right,temp->key);
                }
        return root;
        }

void inorder (pnodeBST root) {
        if (root) {
                inorder(root->left);
                visit(root->key);
                inorder(root->right);
                }
        }

void preorder (pnodeBST root) {
        if (root) {
                visit(root->key);
                preorder(root->left);
                preorder(root->right);
                }
        }

void postorder (pnodeBST root) {
        if (root) {
                postorder(root->left);
                postorder(root->right);
                visit(root->key);
                }
        }

void levelorder (pnodeBST root) {
        bstQueue_ptr levelQueue = createBSTQueue(SIZE);
        pnodeBST EOL = createNode('|');
        pnodeBST EOB = createNode(';');
        pnodeBST blank = createNode('_');
        int ht = height(root);

        enqueueBST(levelQueue,EOL);
        int level = 1, k=0, eob=0;
        for (int i=0; i<=(ht-level); i++)
                printf("\t");
        while (root) {
                if (root!=blank && root!=EOB) {
                        printf("%d\t\t",root->key);
                        if (root->left)
                                enqueueBST(levelQueue,root->left);
                        else
                                enqueueBST(levelQueue,blank);
                        if (root->right)
                                enqueueBST(levelQueue,root->right);
                        else
                                enqueueBST(levelQueue,blank);
                        enqueueBST(levelQueue,EOB);
                        root = dequeueBST(levelQueue);
                        }

                while (root == blank) {
                        root = dequeueBST(levelQueue);
                        printf("\t");
                        k++;
                        }
                while (root == EOB) {
                        root = dequeueBST(levelQueue);
                        if (root != EOL) {
                                printf("  ");
                                eob++;
                                }
                        }
                if (root == EOL) {
                        root = dequeueBST(levelQueue);
                        if (root != NULL) {
                                enqueueBST(levelQueue,EOL);
                                level++;
                                printf("\n\n");
                                for (int i=0; i<=(ht-level); i++)
                                        printf("\t");
                                for (int i=0; i<k; i++)
```

```
					printf("  ");
				for (int i=0; i<eob; i++)
					printf(" ");
				}
			}
		}
	printf("\n");
	}

pnodeBST search (pnodeBST root, int key) {
	if (!root || root->key == key)
		return root;
	if (root->key < key)
		return search(root->right,key);
	return search(root->left,key);
	}

pnodeBST maximum (pnodeBST root) {
	pnodeBST current = root;
	while (current && current->right)
		current = current->right;
	return current;
	}

pnodeBST minimum (pnodeBST root) {
	pnodeBST current = root;
	while (current && current->left)
		current = current->left;
	return current;
	}

void visit (int value) {
	printf("%d ",value);
	}

int height (pnodeBST root) {
	if (!root)
		return 0;
	int leftHt = height(root->left);
	int rightHt = height(root->right);

	if (leftHt > rightHt)
		return leftHt+1;
	else
		return rightHt+1;
	}
```

**bst.c**

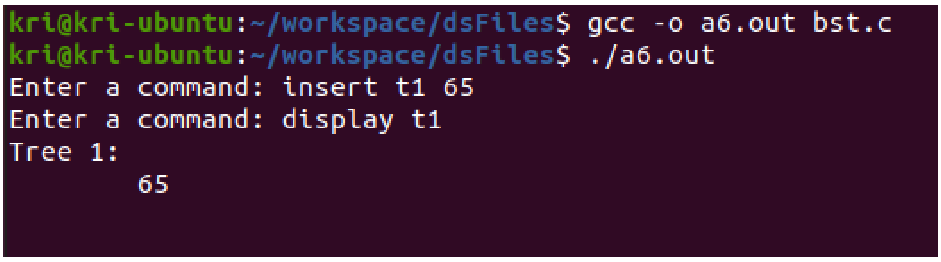
```
//implementation of binary search tree ADT
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bst.h"

int areIdentical(pnodeBST,pnodeBST);
int numLeafNodes(pnodeBST);
int numNonLeafNodes(pnodeBST);
int totalNodes(pnodeBST);

int areIdentical (pnodeBST root1, pnodeBST root2) {
	if (!root1 && !root2)
		return 1;
	if (!root1 || !root2)
		return 0;
	if (root1->key == root2->key)
		return ( areIdentical(root1->left,root2->left) &&
areIdentical(root1->right,root2->right) );
	return 0;
	}

int numLeafNodes (pnodeBST root) {
	if (!root)
		return 0;
	if (!root->left && !root->right)
		return 1;
	int lTotal = numLeafNodes(root->left);
	int rTotal = numLeafNodes(root->right);

	return lTotal+rTotal;
	}

int numNonLeafNodes (pnodeBST root) {
	return totalNodes(root)-numLeafNodes(root);
	}

int totalNodes (pnodeBST root) {
	if (!root)
		return 0;
	if (!root->left && !root->right)
		return 1;

	int lTotal = totalNodes(root->left);
	int rTotal = totalNodes(root->right);

	return lTotal+rTotal+1;
	}

int main() {
	char cmd[20]; int ch=0;
	pnodeBST head[2] = {NULL,NULL};
```

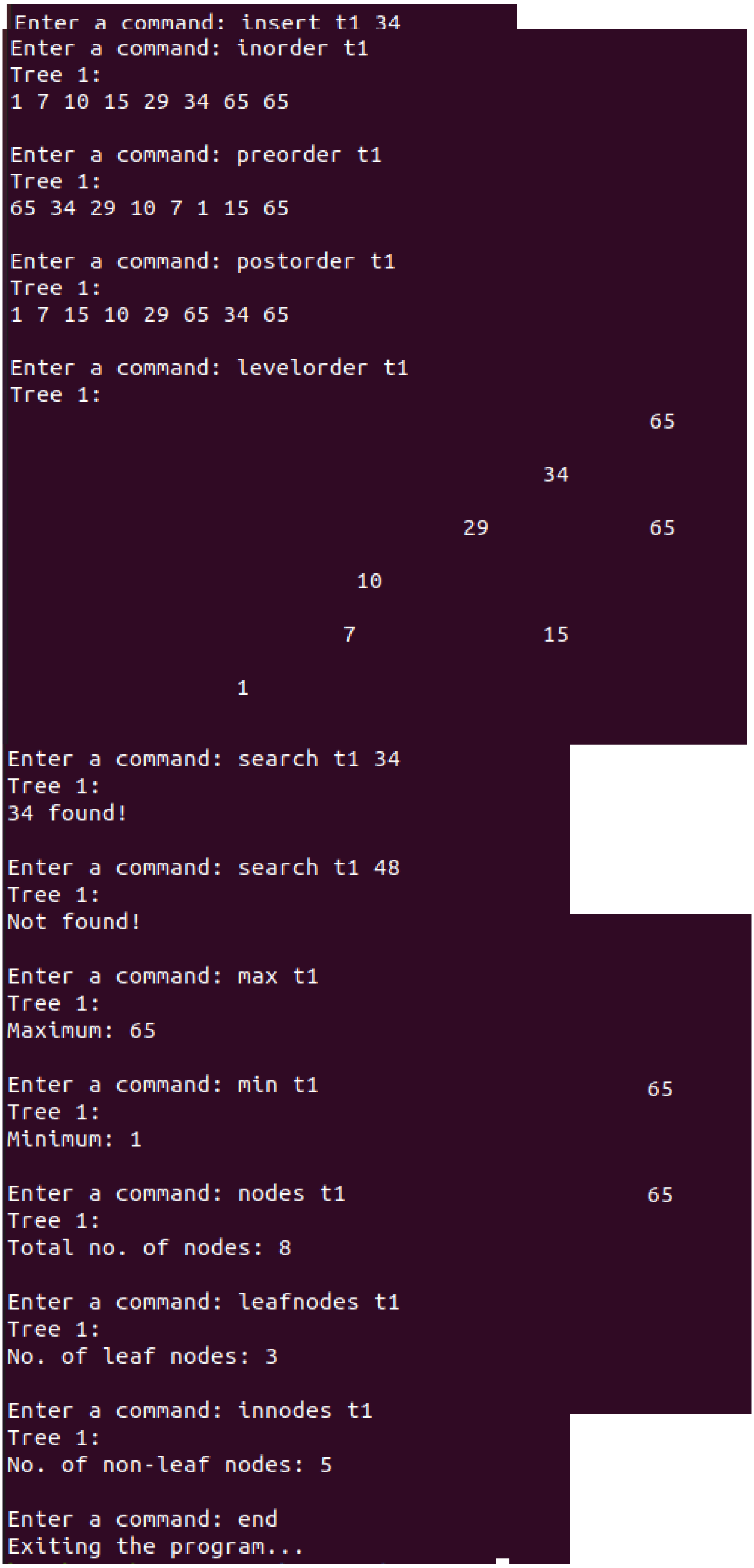
```
printf("Enter a command: ");
scanf("%[^\n]s",cmd);

//use strtok() to split command into 3 parts - instruction, tree, key
int t=0; char* token[3];
token[0] = strtok(cmd," ");
while (token[t] != NULL)
        token[++t] = strtok(NULL," ");

do {
        if (strcmp(token[0],"insert")==0) {
                int value = atoi(token[2]);
                int tree = token[1][strlen(token[1])-1]-'0';
                //printf("Tree %d: \n",tree);
                tree--;
                head[tree] = insertNode(head[tree],value);
                //levelorder(head[tree]);
                }
        else if (strcmp(token[0],"delete")==0) {
                int value = atoi(token[2]);
                int tree = token[1][strlen(token[1])-1]-'0';
                printf("Tree %d: \n",tree);
                tree--;
                head[tree] = deleteNode(head[tree],value);
                levelorder(head[tree]);
                }
        else if (strcmp(token[0],"search")==0) {
                int value = atoi(token[2]);
                int tree = token[1][strlen(token[1])-1]-'0';
                printf("Tree %d: \n",tree);
                tree--;
                pnodeBST temp = search(head[tree],value);
                if (temp)
                        printf("%d found!\n\n",temp->key);
                else
                        printf("Not found!\n\n");
                }
        else if (strcmp(token[0],"max")==0) {
                int tree = token[1][strlen(token[1])-1]-'0';
                printf("Tree %d: \n",tree);
                tree--;
                pnodeBST temp = maximum(head[tree]);
                printf("Maximum: %d\n\n",temp->key);
                }
        else if (strcmp(token[0],"min")==0) {
                int tree = token[1][strlen(token[1])-1]-'0';
                printf("Tree %d: \n",tree);
                tree--;
                pnodeBST temp = minimum(head[tree]);
                printf("Minimum: %d\n\n",temp->key);
                }
        else if (strcmp(token[0],"inorder")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            inorder(head[tree]);
            printf("\n\n");
            }
        else if (strcmp(token[0],"preorder")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            preorder(head[tree]);
            printf("\n\n");
            }
        else if (strcmp(token[0],"postorder")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            postorder(head[tree]);
            printf("\n\n");
            }
        else if (strcmp(token[0],"levelorder")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            levelorder(head[tree]);
            }
        else if (strcmp(token[0],"display")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            levelorder(head[tree]);
            }
        else if (strcmp(token[0],"nodes")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            printf("Total no. of nodes: %d\n\n",totalNodes(head[tree]));
            }
        else if (strcmp(token[0],"leafnodes")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            printf("No. of leaf nodes: %d\n\
n",numLeafNodes(head[tree]));
            }
        else if (strcmp(token[0],"innodes")==0) {
            int tree = token[1][strlen(token[1])-1]-'0';
            printf("Tree %d: \n",tree);
            tree--;
            printf("No. of non-leaf nodes: %d\n\
n",numNonLeafNodes(head[tree]));
            }
```

```
		else if (strcmp(token[0],"identical")==0) {
			int tree1 = token[1][strlen(token[1])-1]-'0';
			int tree2 = token[2][strlen(token[2])-1]-'0';
			tree1--;
			tree2--;
			if (areIdentical(head[tree1],head[tree2]))
				printf("IDENTICAL\n\n");
			else
				printf("NOT IDENTICAL\n\n");
			}
		else if (strcmp(token[0],"end")==0) {
			ch=1;
			printf("Exiting the program...\n");
			exit(0);
			}
		else {
			printf("Invalid command!\n");
			exit(0);
			}

		for (int i=0; i<3; i++) {
			if (token[i] != NULL)
				memset(token[i],0,strlen(token[i]));
			}

		printf("Enter a command: ");
		scanf(" %[^\n]s",cmd);

		t=0;
		token[0] = strtok(cmd," ");
		while (token[t] != NULL)
			token[++t] = strtok(NULL," ");

		} while (ch==0);

	return 0;
	}
```

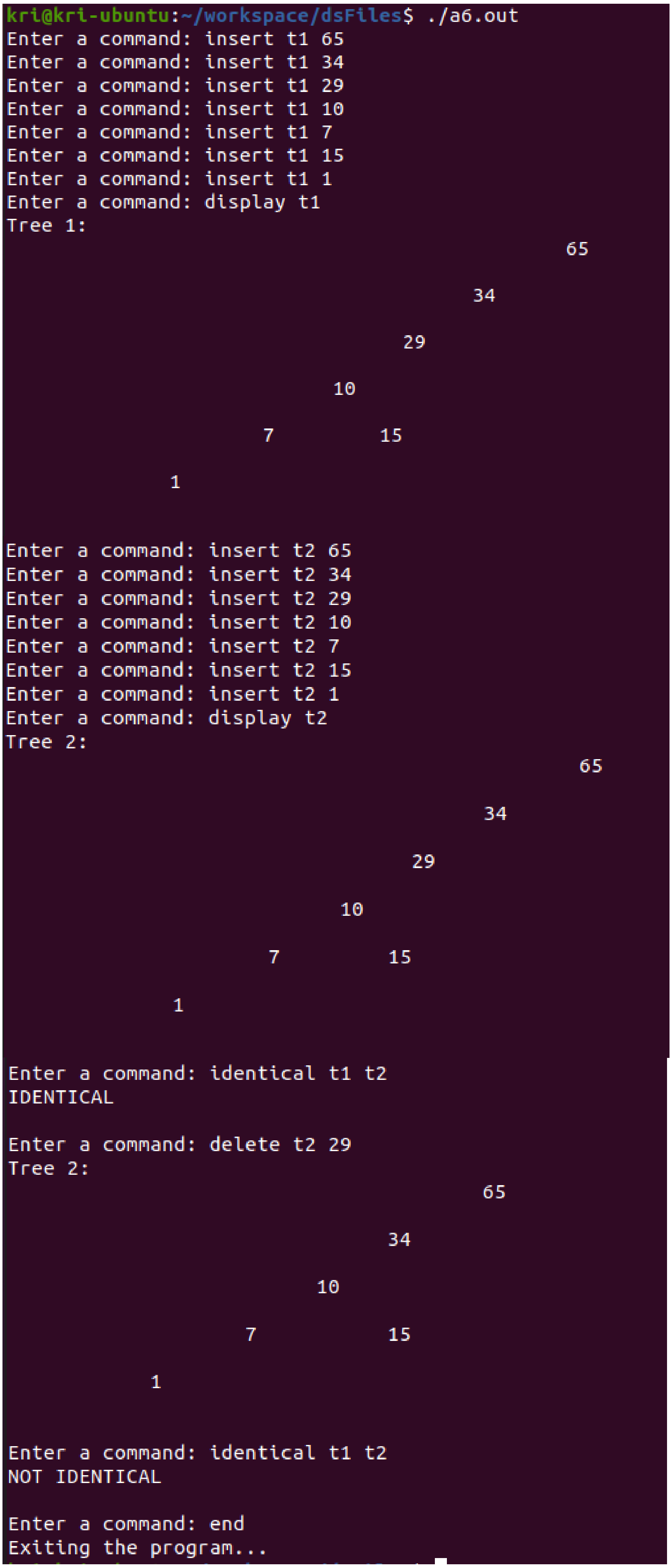
**Output:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a6.out bst.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a6.out
Enter a command: insert t1 65
Enter a command: display t1
Tree 1:
        65
```



```
Enter a command: insert t1 34
Enter a command: inorder t1
Tree 1:
1 7 10 15 29 34 65 65

Enter a command: preorder t1
Tree 1:
65 34 29 10 7 1 15 65

Enter a command: postorder t1
Tree 1:
1 7 15 10 29 65 34 65

Enter a command: levelorder t1
Tree 1:
                                                65

                                        34

                                  29            65

                          10

                         7              15

                 1

Enter a command: search t1 34
Tree 1:
34 found!

Enter a command: search t1 48
Tree 1:
Not found!

Enter a command: max t1
Tree 1:
Maximum: 65

Enter a command: min t1                         65
Tree 1:
Minimum: 1

Enter a command: nodes t1                       65
Tree 1:
Total no. of nodes: 8

Enter a command: leafnodes t1
Tree 1:
No. of leaf nodes: 3

Enter a command: innodes t1
Tree 1:
No. of non-leaf nodes: 5

Enter a command: end
Exiting the program...
```



```
kri@kri-ubuntu:~/workspace/dsFiles$ ./a6.out
Enter a command: insert t1 65
Enter a command: insert t1 34
Enter a command: insert t1 29
Enter a command: insert t1 10
Enter a command: insert t1 7
Enter a command: insert t1 15
Enter a command: insert t1 1
Enter a command: display t1
Tree 1:
                                                65

                                        34

                                  29

                            10

                      7         15

              1


Enter a command: insert t2 65
Enter a command: insert t2 34
Enter a command: insert t2 29
Enter a command: insert t2 10
Enter a command: insert t2 7
Enter a command: insert t2 15
Enter a command: insert t2 1
Enter a command: display t2
Tree 2:
                                                65

                                        34

                                  29

                            10

                      7         15

              1


Enter a command: identical t1 t2
IDENTICAL

Enter a command: delete t2 29
Tree 2:
                                        65

                                32

                          10

                    7           15

            1


Enter a command: identical t1 t2
NOT IDENTICAL

Enter a command: end
Exiting the program...
```

=========================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

================================================================

# A7: Applications of AVL Trees

**Date of submission: 24-11-2021**

**Question:**

1. Create an ADT for a AVL tree (avl.h). (CO2, K3)

   a) Add the following operations: Insert, inorder, search, height

   b) Implement a simple telephone directory manager using AVL Tree Data Structure and provide the following interfaces. Assume that each directory entry holds the name of the person and his telephone number for simplicity. (a7avl.c)

      A. Print_Dir() to print the directory in sorted order of names.

      B. Add_Phone() to include a person and phone number

   c) Demonstrate the binary search tree operations and applications

**Code:**

## avlNode.h

```
//AVL node structure

#define MAX 50

struct AVLNode {
	char data[MAX];
	struct AVLNode *left;
	struct AVLNode *right;
	int height;
	};
typedef struct AVLNode* AVL;

AVL createNode(char[]);

AVL createNode (char item[]) {
	AVL newNode = (AVL) malloc(sizeof(struct AVLNode));
	if (!newNode)
		return NULL; //memory error
	strcpy(newNode->data,item);
	newNode->left = NULL;
	newNode->right = NULL;
	newNode->height = 1;
```

```
        return newNode;
        }
```

**avlQueue.h**

```
//Queue of avl nodes - ADT structure using circular array

#define SIZE 20

struct avlQueue {
        int front;
        int rear;
        int capacity;
        AVL *array;
        };

typedef struct avlQueue* avlQueue_ptr;

avlQueue_ptr createAVLQueue(int);
void enqueueAVL(avlQueue_ptr,AVL);
AVL dequeueAVL(avlQueue_ptr);
int isFullAVL(avlQueue_ptr);
int isEmptyAVL(avlQueue_ptr);

avlQueue_ptr createAVLQueue (int size) {
        avlQueue_ptr treeQueue = (avlQueue_ptr) malloc(sizeof(struct avlQueue));
        treeQueue->capacity = size+1;
        treeQueue->front = -1;
        treeQueue->rear = -1;
        treeQueue->array = (AVL*) malloc(sizeof(struct avlQueue) * treeQueue->capacity);
        return treeQueue;
        }

void enqueueAVL (avlQueue_ptr treeQueue, AVL item) {
        int size = treeQueue->capacity;
        if (isFullAVL(treeQueue))
                return;
        if (isEmptyAVL(treeQueue)) {
                treeQueue->front = 0;
                treeQueue->rear = 0;
                treeQueue->array[treeQueue->rear++] = item;
                return;
                }
        if (treeQueue->rear == size-1 && treeQueue->front != 0) {
                treeQueue->array[treeQueue->rear] = item;
                treeQueue->rear = 0;
                return;
                }
        treeQueue->array[treeQueue->rear++] = item;
```

```
    }

AVL dequeueAVL (avlQueue_ptr treeQueue) {
    if (isEmptyAVL(treeQueue))
        return NULL; //treeQueue empty
    AVL item = treeQueue->array[treeQueue->front];
    if (treeQueue->front == (treeQueue->rear-1)%(treeQueue->capacity)) {
        treeQueue->front = -1;
        treeQueue->rear = -1;
        }
    else if (treeQueue->front == (treeQueue->capacity)-1)
        treeQueue->front = 0;
    else
        treeQueue->front++;
    return item;
    }

int isFullAVL (avlQueue_ptr treeQueue) {
    int size = treeQueue->capacity;
    return (treeQueue->front == 0 && treeQueue->rear == size-1) ||
(treeQueue->rear == (treeQueue->front-1));
    }

int isEmptyAVL (avlQueue_ptr treeQueue) {
    return treeQueue->front == -1;
    }
```

**avl.h**

```
//AVL tree - ADT Structure

#include "avlNode.h"
#include "avlQueue.h"

AVL leftRotate (AVL);
AVL rightRotate (AVL);
AVL insert (AVL,char*);
int search (AVL,char*);
void inorder (AVL);
void levelorder (AVL);
void printEntry (char[]);
int height (AVL);
int max (int,int);

//rotate left
AVL leftRotate (AVL avl1) {
    AVL avl2 = avl1->right;
    avl1->right = avl2->left;
    avl2->left = avl1;

    avl1->height = max(height(avl1->left),height(avl1->right))+1;
    avl2->height = max(height(avl2->left),avl1->height)+1;
```

```
        return avl2;
        }

//rotate right
AVL rightRotate (AVL avl1) {
        AVL avl2=avl1->left;
        avl1->left=avl2->right;
        avl2->right=avl1;

        avl1->height=max(height(avl1->left), height(avl1->right))+1;
        avl2->height=max(height(avl2->left), avl1->height)+1;

        return avl2;
        }

AVL insert (AVL avl, char item[]) {
        if (avl==NULL){
                avl = (AVL) malloc(sizeof(struct AVLNode));
                strcpy(avl->data,item);
                avl->left = avl->right = NULL;
                avl->height = 1;
                return avl;
                }
        if (strcmp(item,avl->data)==0)
                return avl;
        else if (strcmp(item,avl->data)<0) {
                avl->left = insert(avl->left,item);
                if (height(avl->left)-height(avl->right)==2) {
                        if (strcmp(item,avl->left->data)<0)
                                avl = rightRotate(avl);
                        else {
                                avl->left = leftRotate(avl->left);
                                return rightRotate(avl);
                                }
                        }
                }
        else if (strcmp(item,avl->data)>0) {
                avl->right = insert(avl->right,item);
                if (height(avl->right)-height(avl->left)==2) {
                        if (strcmp(item,avl->right->data)>0)
                                avl = leftRotate(avl);
                        else {
                                avl->right = rightRotate(avl->right);
                                return leftRotate(avl);
                                }
                        }
                }
        avl->height = max(height(avl->left),height(avl->right))+1;
        return avl;
        }
```

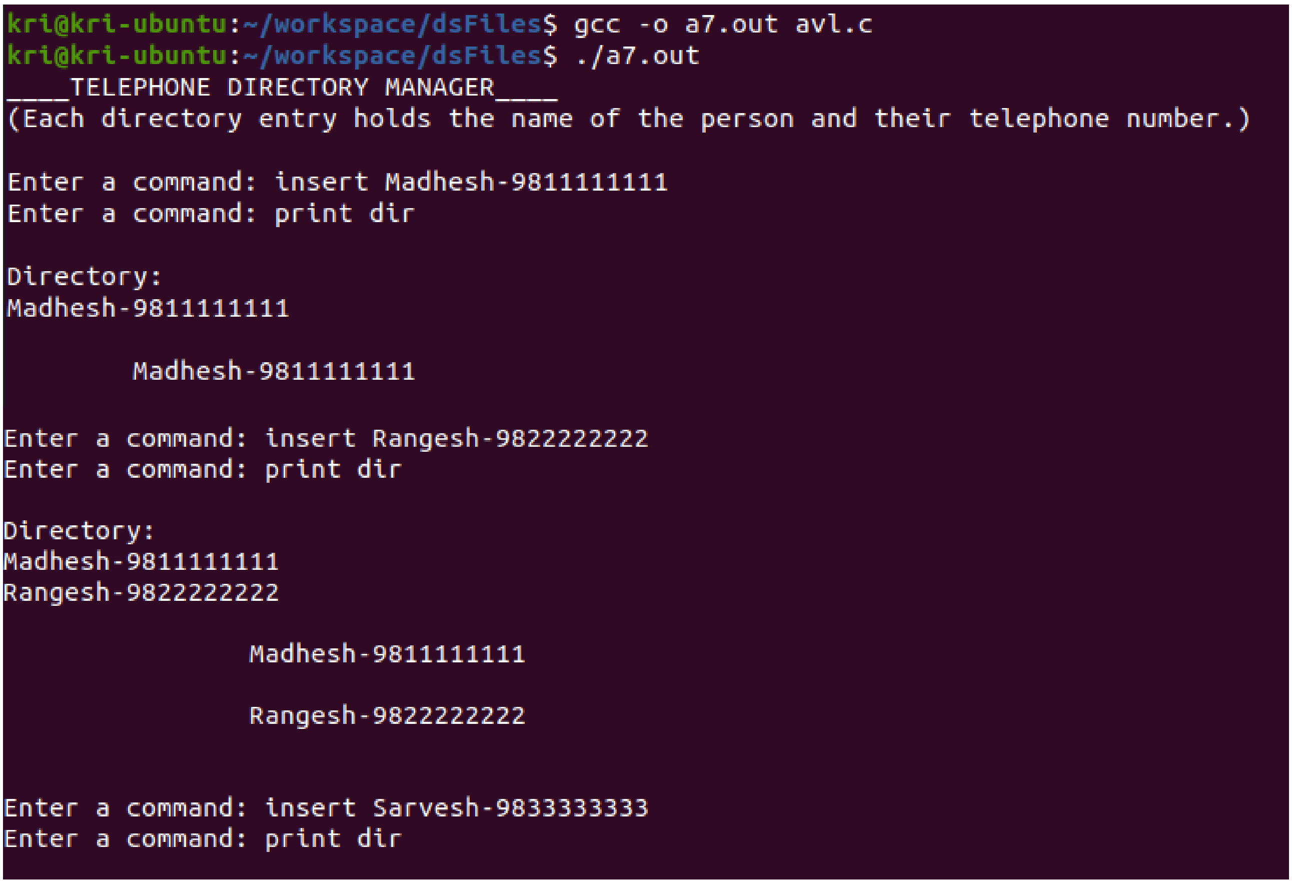
```
int search (AVL avl, char item[]) {
        if (avl) {
                char entry[50], name[50], phone[50];
                char* itemName = strtok(item," ");
                strcpy(entry,avl->data);
                strcpy(name,strtok(entry,"-"));
                strcpy(phone,strtok(NULL," "));

                //to search by name
                if ((strcmp(itemName,name)==0)) {
                        printEntry(avl->data);
                        return 1;
                        }
                else if (strcmp(itemName,name)<0)
                        return search(avl->left,item);
                else
                        return search(avl->right,item);

                }
        return 0;
        }

void inorder (AVL avl) {
        if (avl) {
                inorder(avl->left);
                printf("%s\n",avl->data);
                inorder(avl->right);
                }
        return;
        }

void levelorder (AVL root) {
        avlQueue_ptr levelQueue = createAVLQueue(SIZE);
        AVL EOL = createNode("|");
        AVL EOB = createNode(";");
        AVL blank = createNode("_");
        int ht = height(root);

        enqueueAVL(levelQueue,EOL);
        int level = 1, k=0, eob=0;
        for (int i=0; i<=(ht-level); i++)
                printf("\t");
        while (root) {
                if (root!=blank && root!=EOB) {
                        printf("%s\t\t",root->data);
                        if (root->left)
                                enqueueAVL(levelQueue,root->left);
                        else
                                enqueueAVL(levelQueue,blank);
                        if (root->right)
                                enqueueAVL(levelQueue,root->right);
                        else
```

```
				enqueueAVL(levelQueue,blank);
			enqueueAVL(levelQueue,EOB);
			root = dequeueAVL(levelQueue);
			}

		while (root == blank) {
			root = dequeueAVL(levelQueue);
			printf("\t");
			k++;
			}
		while (root == EOB) {
			root = dequeueAVL(levelQueue);
			if (root != EOL) {
				printf("  ");
				eob++;
				}
			}
		if (root == EOL) {
			root = dequeueAVL(levelQueue);
			if (root != NULL) {
				enqueueAVL(levelQueue,EOL);
				level++;
				printf("\n\n");
				for (int i=0; i<=(ht-level); i++)
					printf("\t");
				for (int i=0; i<k; i++)
					printf("  ");
				for (int i=0; i<eob; i++)
					printf(" ");
				}
			}
		}
	printf("\n");
	}

int height (AVL avl) {
	if (avl==NULL)
		return 0;
	return avl->height;
	}

int max (int a, int b) {
	if (a>b)
		return a;
	return b;
	}

void printEntry (char text[]) {
	printf("Entry: %s\n",text);
	}
```

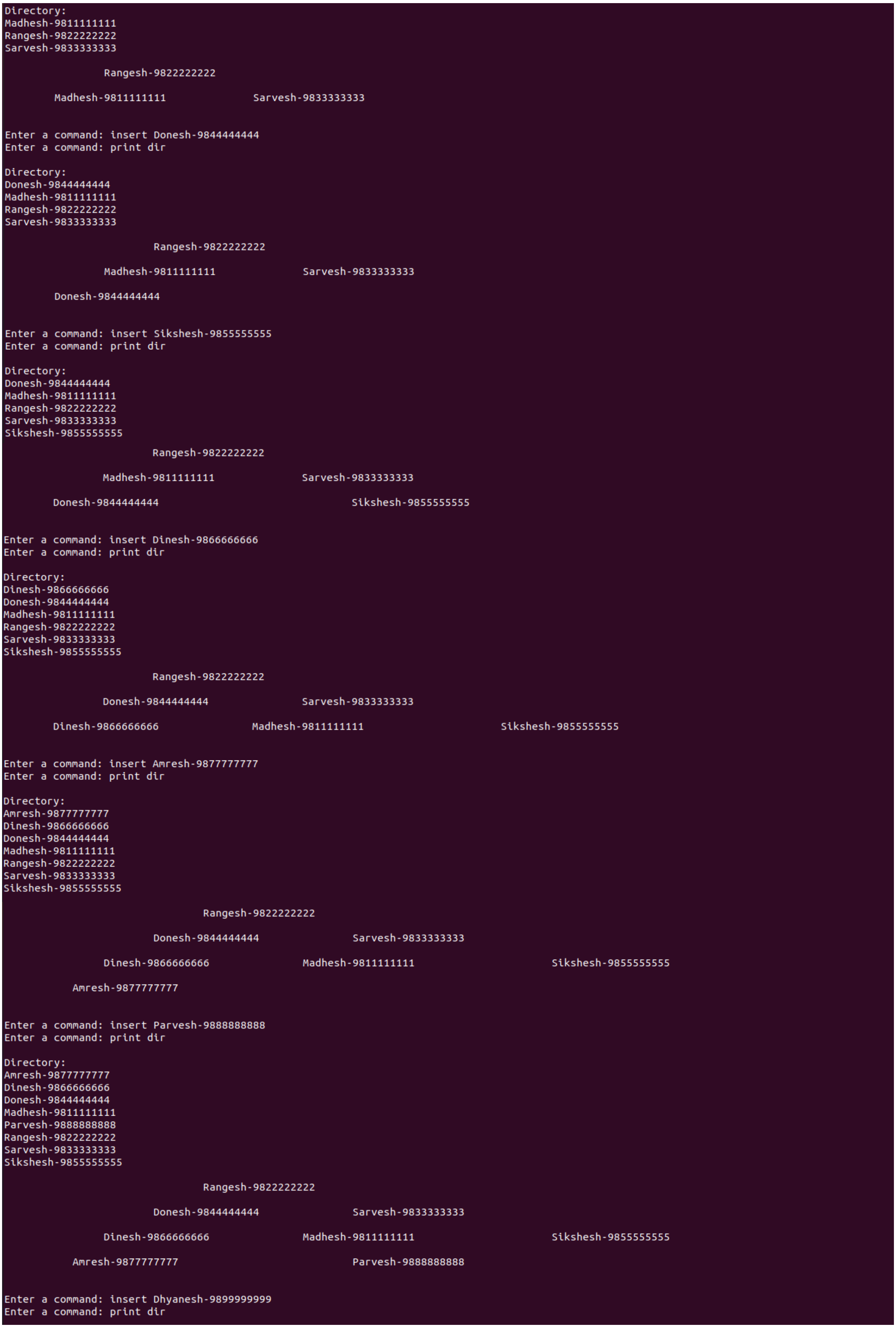
**avl.c**

```
//implementation of AVL tree

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "avl.h"

AVL addPhone (AVL avl, char* item);
void printDir (AVL avl);

int main() {
        printf("____TELEPHONE DIRECTORY MANAGER____\n");
        printf("(Each directory entry holds the name of the person and their
telephone number.)\n\n");

        char cmd[50]; int ch=0;
        AVL avl = NULL;

        //enter command as input from the user
        printf("Enter a command: ");
        scanf("%[^\n]s",cmd);

        //split command into 2 parts - instruction, entry
        char* token[2];
        token[0] = strtok(cmd," ");
        token[1] = strtok(NULL," ");

        do {
                if (strcmp(token[0],"insert")==0) {
                        avl = addPhone(avl,token[1]);
                        //printDir(avl);
                        }
                else if (strcmp(token[0],"print")==0) {
                        printDir(avl);
                        }
                else if (strcmp(token[0],"search")==0) {
                        int result = search(avl,token[1]);
                        if (result == 1)
                                printf("Entry found in directory!\n\n");
                        else if (result == 0)
                                printf("Entry not found in directory!\n\n");
                        }
                else if (strcmp(token[0],"close")==0) {
                        ch=1;
                        printf("Exiting the directory...\n");
                        exit(0);
                        }
                else {
                        printf("Invalid command!\n");
                        exit(0);
                        }

                for (int i=0; i<2; i++)
                        if (token[i] != NULL)
                                memset(token[i],0,strlen(token[i]));

                printf("Enter a command: ");
                scanf(" %[^\n]s",cmd);

                token[0] = strtok(cmd," ");
                token[1] = strtok(NULL," ");

                } while (ch==0);

        return 0;
        }

AVL addPhone (AVL avl, char* item) {
        return insert(avl,item);
        }


void printDir (AVL avl) {
        printf("\nDirectory:\n");
        inorder(avl);
        printf("\n");
        levelorder(avl);
        }
```

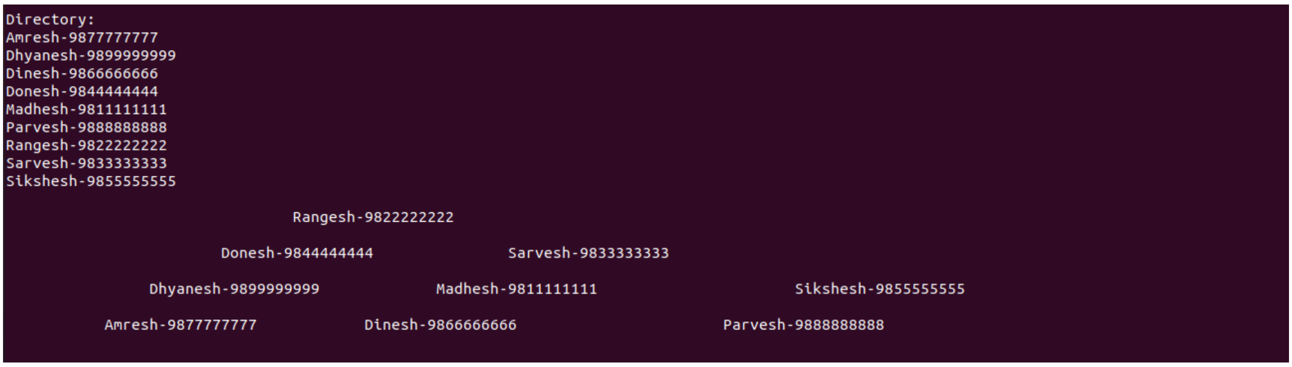
**Output:**



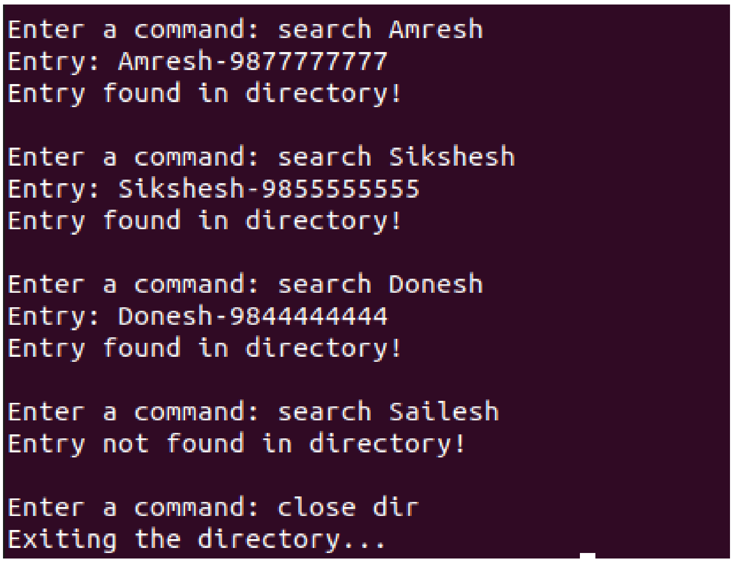
```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a7.out avl.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a7.out
____TELEPHONE DIRECTORY MANAGER____
(Each directory entry holds the name of the person and their telephone number.)

Enter a command: insert Madhesh-9811111111
Enter a command: print dir

Directory:
Madhesh-9811111111

        Madhesh-9811111111

Enter a command: insert Rangesh-9822222222
Enter a command: print dir

Directory:
Madhesh-9811111111
Rangesh-9822222222

                Madhesh-9811111111

                Rangesh-9822222222


Enter a command: insert Sarvesh-9833333333
Enter a command: print dir
```



```
Directory:
Madhesh-9811111111
Rangesh-9822222222
Sarvesh-9833333333

                Rangesh-9822222222

        Madhesh-9811111111              Sarvesh-9833333333


Enter a command: insert Donesh-9844444444
Enter a command: print dir

Directory:
Donesh-9844444444
Madhesh-9811111111
Rangesh-9822222222
Sarvesh-9833333333

                        Rangesh-9822222222

                Madhesh-9811111111              Sarvesh-9833333333

        Donesh-9844444444


Enter a command: insert Sikshesh-9855555555
Enter a command: print dir

Directory:
Donesh-9844444444
Madhesh-9811111111
Rangesh-9822222222
Sarvesh-9833333333
Sikshesh-9855555555

                        Rangesh-9822222222

                Madhesh-9811111111              Sarvesh-9833333333

        Donesh-9844444444                               Sikshesh-9855555555


Enter a command: insert Dinesh-9866666666
Enter a command: print dir

Directory:
Dinesh-9866666666
Donesh-9844444444
Madhesh-9811111111
Rangesh-9822222222
Sarvesh-9833333333
Sikshesh-9855555555

                        Rangesh-9822222222

                Donesh-9844444444               Sarvesh-9833333333

        Dinesh-9866666666               Madhesh-9811111111                      Sikshesh-9855555555


Enter a command: insert Amresh-9877777777
Enter a command: print dir

Directory:
Amresh-9877777777
Dinesh-9866666666
Donesh-9844444444
Madhesh-9811111111
Rangesh-9822222222
Sarvesh-9833333333
Sikshesh-9855555555

                                Rangesh-9822222222

                        Donesh-9844444444               Sarvesh-9833333333

                Dinesh-9866666666               Madhesh-9811111111                      Sikshesh-9855555555

           Amresh-9877777777


Enter a command: insert Parvesh-9888888888
Enter a command: print dir

Directory:
Amresh-9877777777
Dinesh-9866666666
Donesh-9844444444
Madhesh-9811111111
Parvesh-9888888888
Rangesh-9822222222
Sarvesh-9833333333
Sikshesh-9855555555

                                Rangesh-9822222222

                        Donesh-9844444444               Sarvesh-9833333333

                Dinesh-9866666666               Madhesh-9811111111                      Sikshesh-9855555555

           Amresh-9877777777                            Parvesh-9888888888


Enter a command: insert Dhyanesh-9899999999
Enter a command: print dir
```



```
Directory:
Amresh-9877777777
Dhyanesh-9899999999
Dinesh-9866666666
Donesh-9844444444
Madhesh-9811111111
Parvesh-9888888888
Rangesh-9822222222
Sarvesh-9833333333
Sikshesh-9855555555

                                Rangesh-9822222222

                        Donesh-9844444444               Sarvesh-9833333333

                Dhyanesh-9899999999             Madhesh-9811111111                      Sikshesh-9855555555

           Amresh-9877777777            Dinesh-9866666666                       Parvesh-9888888888
```



```
Enter a command: search Amresh
Entry: Amresh-9877777777
Entry found in directory!

Enter a command: search Sikshesh
Entry: Sikshesh-9855555555
Entry found in directory!

Enter a command: search Donesh
Entry: Donesh-9844444444
Entry found in directory!

Enter a command: search Sailesh
Entry not found in directory!

Enter a command: close dir
Exiting the directory...
```

==============================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

==============================================================

## A8: Priority Queue – Application of Binary Heap

**Date of submission: 24-11-2021**

**Question:**

1. Create an ADT for a binary heap (heap.h). (CO2, K3)

a) Add the following operations: buildHeap, Insert, deleteMin

b) Implement a priority Queue of beneficiaries of below poverty line families based on their income. (a8PQ.c). It is required to find beneficiaries below the specified limit of the salary. Write an appropriate function.

**Code:**

### binHeap.h

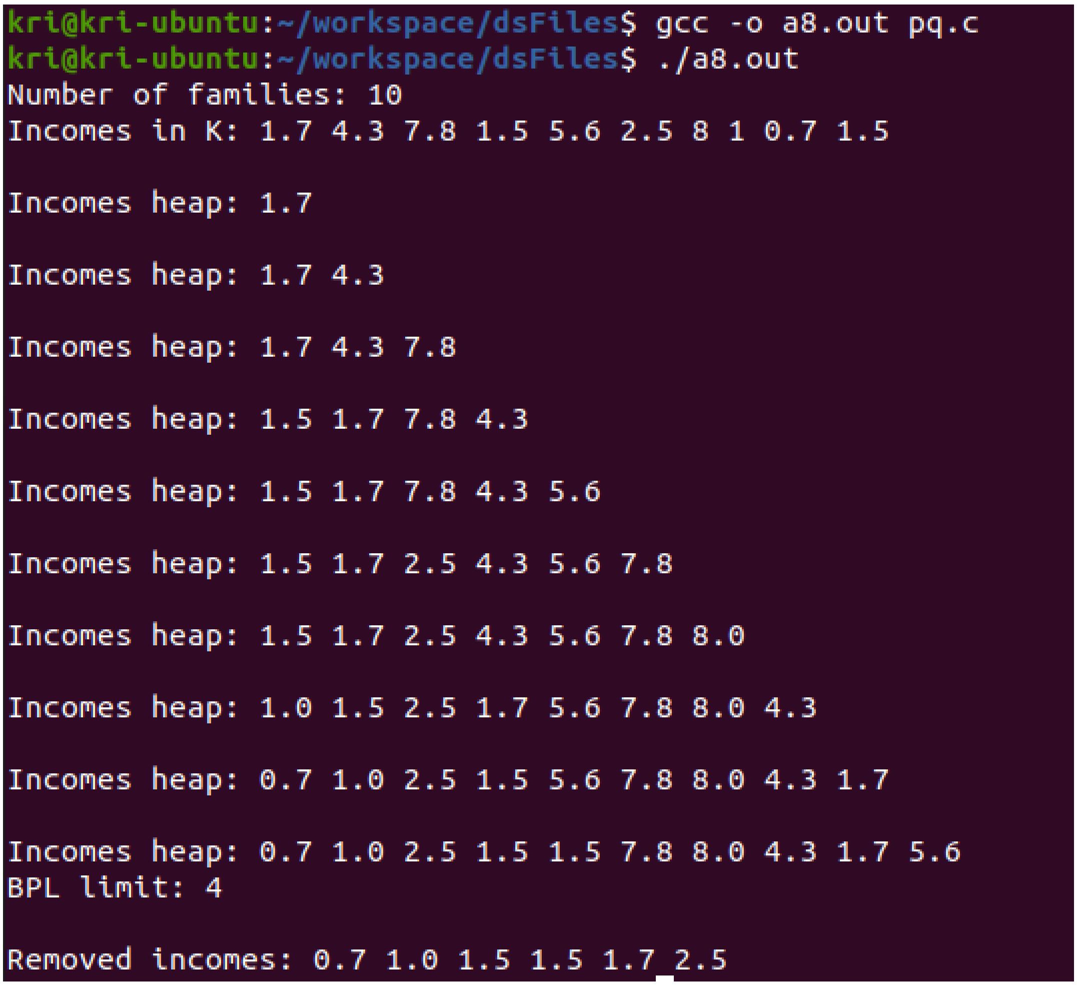
```
//Binary Heap - MinHeap - ADT Structure using arrays
#define EMPTY -1

struct Heap {
	unsigned capacity;
	int size;
	float* arr;
	};
typedef struct Heap* minHeap;

minHeap buildHeap (unsigned);
minHeap insert (minHeap,float);
float deleteMin (minHeap);
int parent (int);
int leftChild (int);
int rightChild (int);
void swap (float[],int,int);
void heapifyUp (minHeap,int);
void heapifyDown (minHeap,int);

minHeap buildHeap (unsigned capacity) {
	minHeap heap = (minHeap) malloc (sizeof(struct Heap));
	heap->capacity = capacity;
	heap->size = -1;
	heap->arr = (float*) malloc (heap->capacity * sizeof(float));
```

```
        return heap;
        }

minHeap insert (minHeap heap, float key) {
        //if heap is full
        if (heap->size!=-1 && (heap->size >= heap->capacity-1))
                return heap;
        //insert key into heap as last node
        heap->arr[++heap->size] = key;
        //ensure heap structure
        heapifyUp(heap,heap->size);
        return heap;
        }

float deleteMin (minHeap heap) {
        //if heap is empty
        if (heap->size == -1)
                return EMPTY;
        //swap root node key and last node key
        swap(heap->arr,0,heap->size);
        //delete last node
        float temp = heap->arr[heap->size--];
        //ensure heap structure
        heapifyDown(heap,0);
        //return minimum key
        return temp;
        }

int parent (int i) {
        return (i-1)/2;
        }

int leftChild (int i) {
        return (2*i)+1;
        }

int rightChild (int i) {
        return (2*i)+2;
        }

void swap (float heapArr[], int pos1, int pos2) {
        float temp = heapArr[pos1];
        heapArr[pos1] = heapArr[pos2];
        heapArr[pos2] = temp;
        }

void heapifyUp (minHeap heap, int i) {
        while (i>0 && (heap->arr[parent(i)] > heap->arr[i])) {
                //swap parent node key and current node key
                swap(heap->arr,parent(i),i);
                //update i to parent of i
                i = parent(i);
                }
        }

void heapifyDown (minHeap heap, int i) {
        int minIndex = i;
        //left child check
        int l = leftChild(i);
        if ((l <= heap->size) && (heap->arr[l] < heap->arr[minIndex]))
                minIndex = l;
        //right child check
        int r = rightChild(i);
        if ((r <= heap->size) && (heap->arr[r] < heap->arr[minIndex]))
                minIndex = r;
        //swap minimum key with current node key
        if (i!=minIndex) {
                swap(heap->arr,minIndex,i);
                heapifyDown(heap,minIndex);
                }
        }
```

**pq.c**

```
//Priority queue - Application of Binary Heap

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

void display (minHeap);

int main() {
        int nFam;
        printf("Number of families: ");
        scanf("%d",&nFam);
        minHeap incomes = NULL;
        incomes = buildHeap(nFam);

        float value;
        printf("Incomes in K: ");
        for (int i=0; i<nFam; i++) {
                scanf("%f",&value);
                incomes = insert(incomes,value);
                display(incomes);
                }

        float limit;
        printf("BPL limit: ");
        scanf("%f",&limit);

        printf("\nRemoved incomes: ");
        for (int i=0; incomes->arr[0]<=4 ; i++) {
                float del = deleteMin(incomes);
                printf("%.1f ",del);
                }
        printf("\n");

        return 0;
        }

void display (minHeap heap) {
        printf("\nIncomes heap: ");
        for (int i=0; i<=heap->size; i++)
                printf("%.1f ",heap->arr[i]);
        printf("\n");
        }
```

**Output:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a8.out pq.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a8.out
Number of families: 10
Incomes in K: 1.7 4.3 7.8 1.5 5.6 2.5 8 1 0.7 1.5

Incomes heap: 1.7

Incomes heap: 1.7 4.3

Incomes heap: 1.7 4.3 7.8

Incomes heap: 1.5 1.7 7.8 4.3

Incomes heap: 1.5 1.7 7.8 4.3 5.6

Incomes heap: 1.5 1.7 2.5 4.3 5.6 7.8

Incomes heap: 1.5 1.7 2.5 4.3 5.6 7.8 8.0

Incomes heap: 1.0 1.5 2.5 1.7 5.6 7.8 8.0 4.3

Incomes heap: 0.7 1.0 2.5 1.5 5.6 7.8 8.0 4.3 1.7

Incomes heap: 0.7 1.0 2.5 1.5 1.5 7.8 8.0 4.3 1.7 5.6
BPL limit: 4

Removed incomes: 0.7 1.0 1.5 1.5 1.7 2.5
```

==========================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** |  |  |  |

==================================================================

## A9: Learning to implement and traverse Graphs and their Applications

**Date of submission: 15-12-2021**

**Question:**

1. Create a graph as an adjacency matrix and traverse it using DFS (recursive) and BFS (iterative).
2. Test bfs() for a disconnected graph
3. You are given a set of persons P and their friendship relation R. That is, (a, b) ∈ R iff a is a friend of b. You must find a way to introduce person x to person y through a chain of friends. Model this problem with a graph and solve the problem.
4. You are given a set of cities and aircrafts routes. Model the routes using a graph. Given two cities, find whether there exists a direct flight between two cities; if not use a search algorithm and show the hops and connecting flights.

**Code:**

### queue.h

```
//Queue ADT structure using circluar array

struct Queue {
	int front;
	int rear;
	int capacity;
	int *array;
	};

typedef struct Queue* queue_ptr;

queue_ptr createQueue(int);
void enqueue(queue_ptr,int);
int dequeue(queue_ptr);
int isFull(queue_ptr);
int isEmpty(queue_ptr);
int getRear(queue_ptr);
int getFront(queue_ptr);

queue_ptr createQueue (int size) {
	queue_ptr queue = (queue_ptr) malloc(sizeof(struct Queue));
	queue->capacity = size+1;
```

```
        queue->front = -1;
        queue->rear = -1;
        queue->array = (int*) malloc(sizeof(int) * queue->capacity);
        return queue;
        }

void enqueue (queue_ptr queue, int item) {
        int size = queue->capacity;
        if (isFull(queue))
                return; //queue full
        if (isEmpty(queue)) {
                queue->front = 0;
                queue->rear = 0;
                queue->array[queue->rear++] = item;
                return;
                }
        if (queue->rear == size-1 && queue->front != 0) {
                queue->array[queue->rear] = item;
                queue->rear = 0;
                return;
                }
        queue->array[queue->rear++] = item;
        }

int dequeue (queue_ptr queue) {
        if (isEmpty(queue))
                return -1; //queue empty
        int item = queue->array[queue->front];
        if (queue->front == (queue->rear-1)%(queue->capacity)) {
                queue->front = -1;
                queue->rear = -1;
                }
        else if (queue->front == (queue->capacity)-1)
                queue->front = 0;
        else
                queue->front++;
        return item;
        }

int isFull (queue_ptr queue) {
        int size = queue->capacity;
        return (queue->front == 0 && queue->rear == size-1) || (queue->rear
== (queue->front-1)); //% needed?
        }

int isEmpty (queue_ptr queue) {
        return queue->front == -1;
        }

int getRear (queue_ptr queue) {
        return queue->array[queue->rear];
        }
```

```
int getFront (queue_ptr queue) {
        return queue->array[queue->front];
        }
```

**graph.h**

```
//Graph ADT Structure

#define MAX_SIZE 20
#include "queue.h"

struct Graph {
        int adj[MAX_SIZE][MAX_SIZE];
        int size;
        int visited[MAX_SIZE];
        };
typedef struct Graph* graph;

graph createGraph (int);
void dfs (graph,int);
void bfs (graph,int);
void printbfs (graph,int);
int bfsSearch (graph,int,int);
void dfsSearch (graph,int,int);
void clearVisited(graph);
void initVertices (graph,int);
void insertEdgeD (graph,int,int);
void insertEdge (graph,int,int);

graph createGraph (int numVertices) {
        graph newGraph = (graph) malloc (sizeof(struct Graph));
        newGraph->size = numVertices;
        for (int i=0; i<numVertices; i++) {
                newGraph->visited[i] = 0;
                for (int j=0; j<numVertices; j++)
                        newGraph->adj[i][j] = 0;
                }
        return newGraph;
        }

void dfs (graph G, int start) {
        int vertex = start;
        G->visited[vertex] = 1;
        printf("%d ",vertex+1);
        for (int i=1; i<=G->size; i++) {
                if (G->adj[vertex][i] && !G->visited[i]) {
                        dfs(G,i);
                        }
                }
        }
```

```
void bfs (graph G, int start) {
        int vertex = start;
        queue_ptr queue = createQueue(MAX_SIZE);

        enqueue(queue,vertex);
        G->visited[vertex] = 1;

        while (!isEmpty(queue)) {
                int u = dequeue(queue);
                printf("%d ", u+1);
                for (int i=0; i<G->size; i++) {
                        if (G->adj[u][i] && !G->visited[i]) {
                                G->visited[i] = 1;
                                enqueue(queue,i);
                                }
                        }
                }
        }

void dfsSearch (graph G, int start, int dest) {
        int vertex = start;
        G->visited[vertex] = 1;
        printf("%d ",vertex+1);
        if (vertex == dest)
                return;
        for (int i=1; i<=G->size; i++) {
                if (G->adj[vertex][i] && !G->visited[i]) {
                        dfs(G,i);
                        }
                }
        }

void printbfs (graph G, int start) {
        int vertex = start;
        queue_ptr queue = createQueue(MAX_SIZE);
        clearVisited(G);

        enqueue(queue,vertex);
        G->visited[vertex] = 1;

        while (!isEmpty(queue)) {
                int u = dequeue(queue);
                printf("%d ", u+1);
                G->visited[u]==2;
                for (int i=0; i<G->size; i++) {
                        if (G->adj[u][i]==1 && G->visited[i]==0) {
                                G->visited[i] = 1;
                                enqueue(queue,i);
                                }
                        }
                }
        for (int v=0; v<G->size; v++) {
                if (!G->visited[v]) {
                        printf("\n");
                        vertex = v;

                        enqueue(queue,vertex);
                        G->visited[vertex] = 1;

                        while (!isEmpty(queue)) {
                                int u = dequeue(queue);
                                printf("%d ", u+1);
                                G->visited[u]==2;
                                for (int i=0; i<G->size; i++) {
                                        if (G->adj[u][i]==1 && G->visited[i]==0) {
                                                G->visited[i] = 1;
                                                enqueue(queue,i);
                                                }
                                        }
                                }
                        }
                }
        }

int bfsSearch (graph G, int start, int dest) {
        int vertex = start;
        queue_ptr queue = createQueue(MAX_SIZE);

        clearVisited(G);

        enqueue(queue,vertex);
        G->visited[vertex] = 1;

        while (!isEmpty(queue)) {
                int u = dequeue(queue);
                printf("%d ", u+1);
                G->visited[u]==2;

                if (u==dest)
                        return 1;

                for (int i=0; i<G->size; i++) {
                        if (G->adj[u][i]==1 && G->visited[i]==0) {
                                G->visited[i] = 1;
                                enqueue(queue,i);
                                }
                        }
                }
        return -1;
        }

void clearVisited (graph G) {
        for (int i=0; i<G->size; i++)
```

```
                G->visited[i]=0;
        }

void initVertices (graph G, int size) {
        G->size = size;
        }

void insertEdgeD (graph G, int vertex1, int vertex2) {
        G->adj[vertex1][vertex2] = 1;
        }

void insertEdge (graph G, int vertex1, int vertex2) {
        G->adj[vertex1][vertex2] = 1;
        G->adj[vertex2][vertex1] = 1;
        }
```

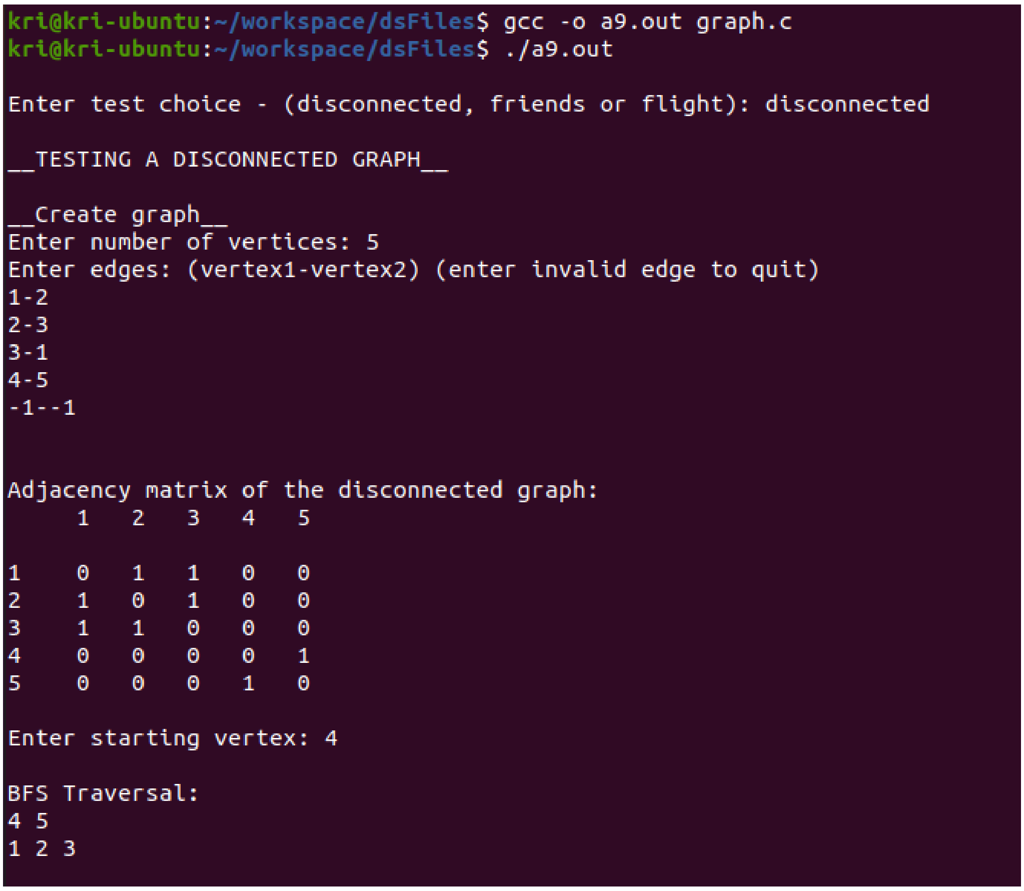
**graph.c**

```
//implementation of graph ADT - bfs and dfs

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "graph.h"

struct City {
        char name[MAX_SIZE][MAX_SIZE];
        int code;
        };
typedef struct City* city;

void dfsFlight (graph,city,int,int);
int bfsFlight (graph,int,int);
int getIndex (city,char*);
void printAdjMatrix (graph);
void printAdjMatrixFlight (graph);
int reached;

int main() {
        char cmd[50];

        do {
                printf("\nEnter test choice - (disconnected, friends or flight): ");
                scanf("%s",cmd);

                if (strcmp(cmd,"disconnected")==0) {
                        printf("\n__TESTING A DISCONNECTED GRAPH__\n");

                        int numV, numE, maxE;

                        printf("\n__Create graph__\n");
                        printf("Enter number of vertices: ");
```

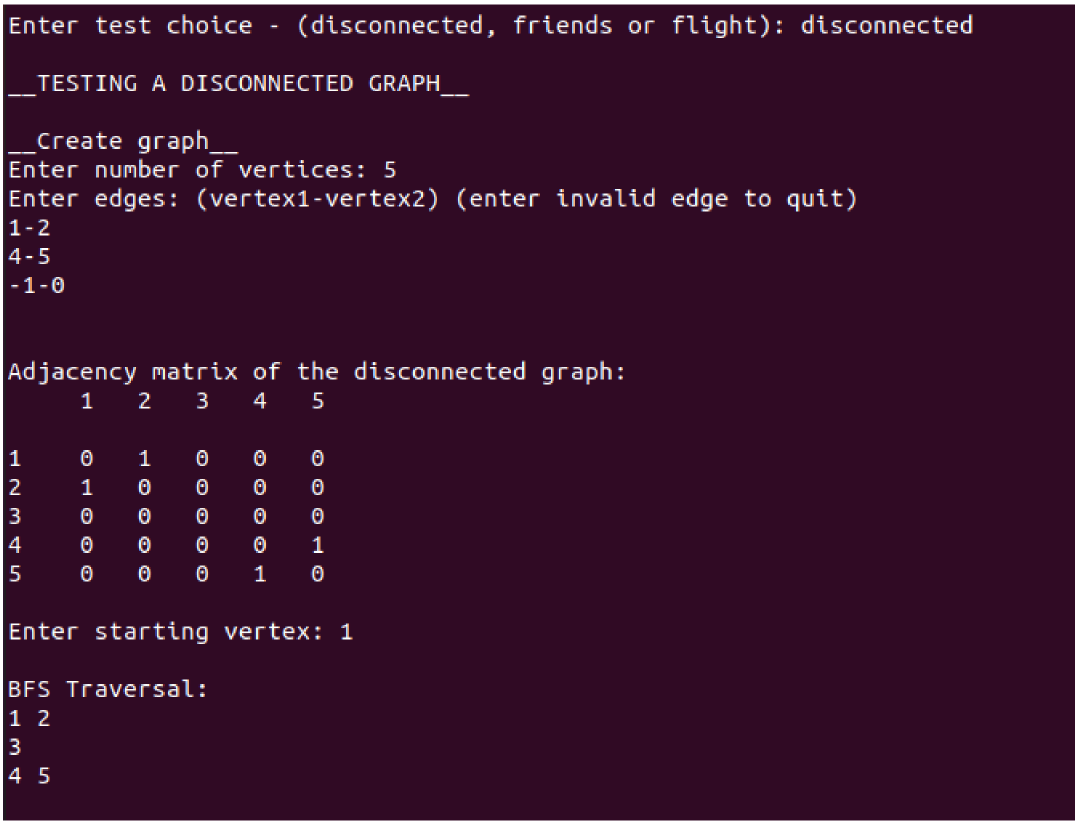
```
            scanf("%d",&numV);
            graph g1 = createGraph(numV);

            maxE = numV*(numV-1)/2;

            printf("Enter edges: (vertex1-vertex2) (enter invalid edge to
quit)\n");
            int v1,v2;
            numE=0;
            do {
                scanf("%d-%d",&v1,&v2);
                if (v1>0 && v2>0) {
                    insertEdge(g1,v1-1,v2-1);
                    numE++;
                    }
                } while(numE<=maxE && (v1>0 && v2>0));

            printf("\n\nAdjacency matrix of the disconnected graph:");
            printAdjMatrix(g1);

            int start;
            printf("Enter starting vertex: ");
            scanf("%d",&start);

            //printf("\nDFS Traversal: \n");
            //dfs(g1,start-1);
            printf("\nBFS Traversal: \n");
            printbfs(g1,start-1);
            printf("\n\n");
            }

        else if (strcmp(cmd,"friends")==0) {
            printf("\n__FRIENDSHIP RELATION__\n");

            int numV, numE, maxE;

            printf("\n__Create graph with people as vertices and
relations given by edges__\n");
            printf("Enter number of people: ");
            scanf("%d",&numV);
            graph g2 = createGraph(numV);

            maxE = numV*(numV-1)/2;

            printf("Enter pair of friends: (vertex1-vertex2) (enter invalid
friendship to quit)\n");
            int v1,v2;
            numE=0;
            do {
                scanf("%d-%d",&v1,&v2);
                if (v1>0 && v2>0) {
                    insertEdge(g2,v1-1,v2-1);
                    numE++;
                    }
                } while(numE<=maxE && (v1>0 && v2>0));

            printf("\n\nAdjacency matrix of the friendship graph:");
            printAdjMatrix(g2);

            int x, y;
            printf("Enter pair of people to introduce: ");
            scanf("%d %d", &x, &y);

            //printf("BFS Traversal: ");
            //printbfs(g2,x-1);
            printf("\n\n");

            int find = bfsSearch(g2,x-1,y-1);
            clearVisited(g2);
            //dfsSearch(g2,x-1,y-1);
            if (find == -1)
                printf("\nThe given pair of people cannot be
introduced!\n");
            else
                printf("\nThey can be introduced!\n");
            printf("\n");
            }

        else if (strcmp(cmd,"flight")==0) {
            printf("\n__FLIGHT ROUTES__\n");

            city C = (city) malloc(sizeof(struct City));
            C->code = 0;

            char flight1[MAX_SIZE], flight2[MAX_SIZE];
            int numV, numE, maxE;

            printf("\n__Create graph with flight schedule__\n");
            printf("Enter number of cities: ");
            scanf("%d",&numV);
            graph flights = createGraph(numV);

            maxE = numV*(numV-1)/2;

           printf("Enter direct flights: (vertex1 vertex2) (enter nil nil to
quit)\n");
            int v1,v2;
            numE=0;
            do {
                printf("Enter the number of direct flights: ");
                scanf("%d",&numE);
                if (numE > maxE)
                    printf("Number of direct flights cannot exceed
%d. Enter again.",maxE);
```

```
                } while (numE>maxE);

                printf("Enter direct flights:\n");
                for (int i=0; i<numE; i++) {
                    int f1, f2;
                    scanf("%s %s", flight1,flight2);
                    f1 = getIndex(C,flight1);
                    f2 = getIndex(C,flight2);
                    insertEdgeD(flights,f1,f2);
                    }

                printAdjMatrixFlight(flights);

                char src[MAX_SIZE], dest[MAX_SIZE];

                printf("Enter pair of cities to check for direct flights: ");
                memset(flights->visited,0,sizeof(flights->visited));
                scanf("%s %s",src,dest);

                while (strcmp(src,"nil") && strcmp(dest,"nil")) {
                    reached = 0;
                    if (getIndex(C,src) <= flights->size && getIndex(C,dest) <= flights->size) {
                        int s = getIndex(C,src), d = getIndex(C,dest);
                        if (flights->adj[s][d])
                            printf("\nDirect flight available: %s -> %s\n",C->name[s],C->name[d]);
                        else {
                            clearVisited(flights);

                            printf("Checking for connecting flights\n");

                            dfsFlight(flights,C,s,d);

                            }
                        }
                    else
                        printf("City not found\n");
                    printf("\n");

                    printf("Enter pair of cities to check for direct flights: ");
                    clearVisited(flights);
                    scanf("%s %s",src,dest);
                    }
            }

        } while (strcmp(cmd,"exit"));
    }

void dfsFlight (graph G, city C, int source, int dest) {
        G->visited[source]++;
        if (source != dest && reached==0)
                printf("%s -> ", C->name[source]);
        if (source == dest) {
                printf("%s \n", C->name[source]);
                reached = 1;
                return;
                }
        for (int i = 1; i <= G->size; i ++) {
        if (i!=source) {
                        if (G->adj[source][i] && G->visited[i]<2) {
                                dfsFlight(G,C,i,dest);
                                }
                        }
        }
        return;
        }

int bfsFlight (graph G, int vertex, int dest) {
        queue_ptr queue = createQueue(MAX_SIZE);

        enqueue(queue,vertex);
        G->visited[vertex] = 1;

        while (!isEmpty(queue)) {
                int u = dequeue(queue);
                if (u==dest)
                        return 1;
                for (int i=1; i<=G->size; i++) {
                        if (G->adj[u][i] && !G->visited[i]) {
                                G->visited[i] = 1;
                                enqueue(queue, i);
                                }
                        }
                }
        return -1;
        }

int getIndex (city C, char* findCity) {
        for (int i=1; i<=C->code; i++) {
                if (!strcmp(C->name[i],findCity))
                        return i;
                }
        strcpy(C->name[++C->code],findCity);
        return C->code;
        }

void printAdjMatrix (graph G) {
        int size = G->size;
        printf("\n     ");
        for (int i=1; i<=size; i++)
                printf("%d   ",i);
        printf("\n\n");
        for (int i=0; i<size; i++) {
                printf("%d     ",i+1);
                for (int j=0; j<size; j++) {
                        printf("%d    ",G->adj[i][j]);
                        }
                printf("\n");
                }
        printf("\n");
        }

void printAdjMatrixFlight (graph G) {
        int size = G->size;
        printf("\n      ");
        for (int i=1; i<=size; i++)
                printf("%d    ",i);
        printf("\n\n");
        for (int i=1; i<=size; i++) {
                printf("%d     ",i);
                for (int j=1; j<=size; j++) {
                        printf("%d    ",G->adj[i][j]);
                        }
                printf("\n");
                }
        printf("\n");
        }
```

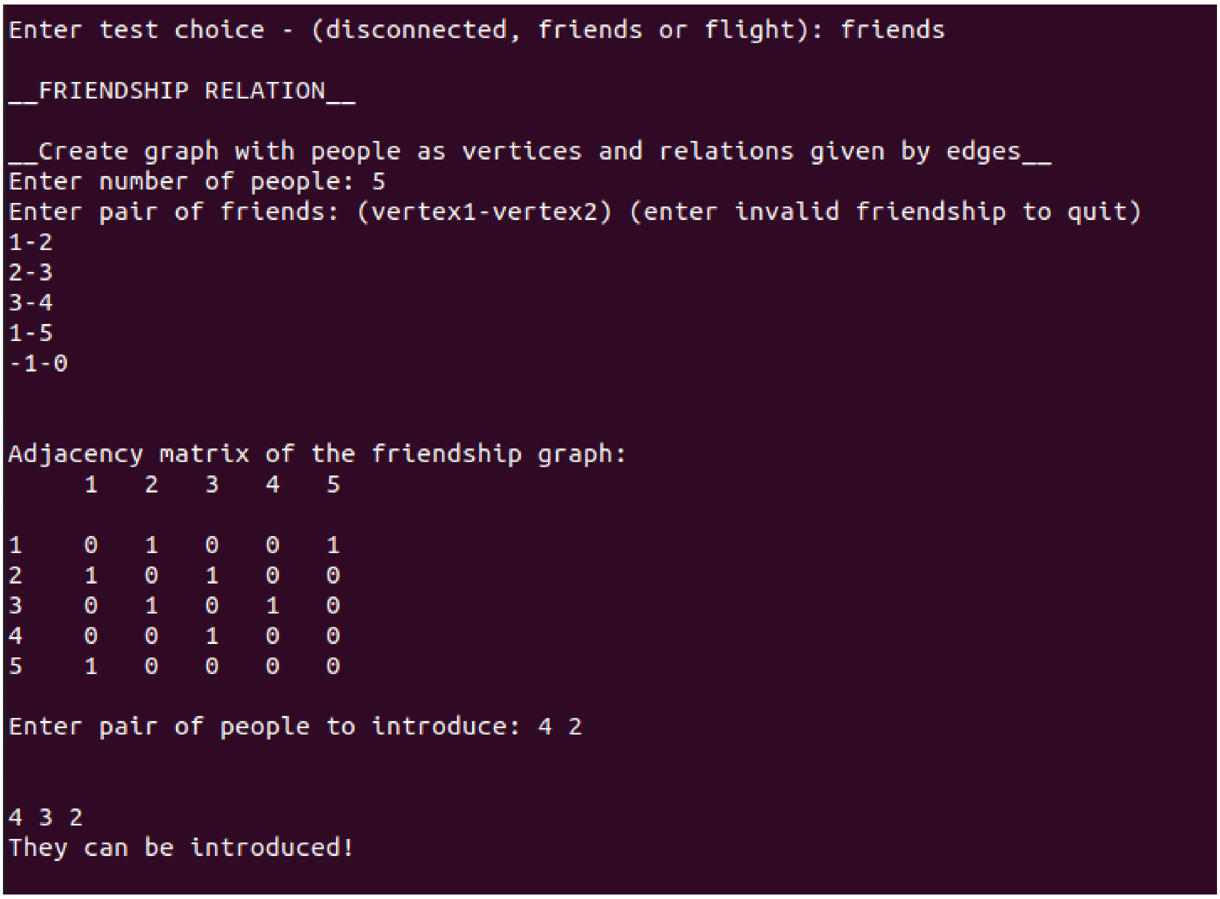
**Output:**



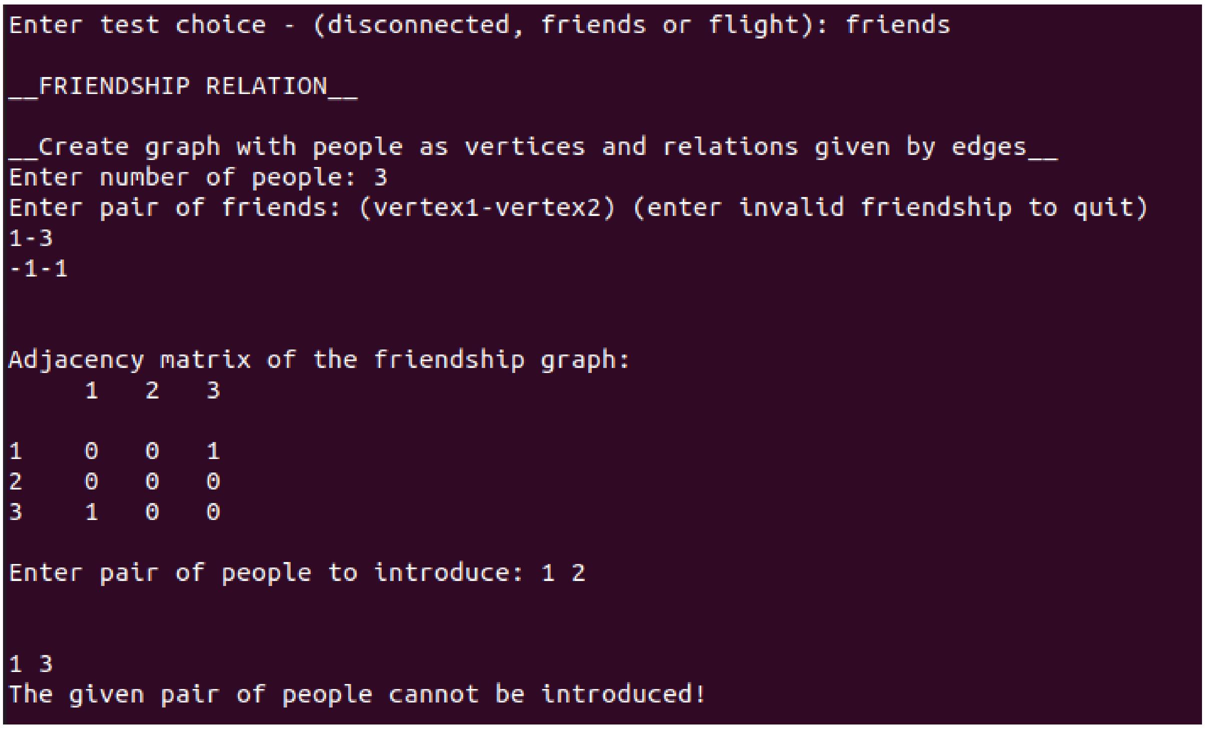
```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a9.out graph.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a9.out

Enter test choice - (disconnected, friends or flight): disconnected

__TESTING A DISCONNECTED GRAPH__

__Create graph__
Enter number of vertices: 5
Enter edges: (vertex1-vertex2) (enter invalid edge to quit)
1-2
2-3
3-1
4-5
-1--1


Adjacency matrix of the disconnected graph:
     1   2   3   4   5

1    0   1   1   0   0
2    1   0   1   0   0
3    1   1   0   0   0
4    0   0   0   0   1
5    0   0   0   1   0

Enter starting vertex: 4

BFS Traversal:
4 5
1 2 3
```



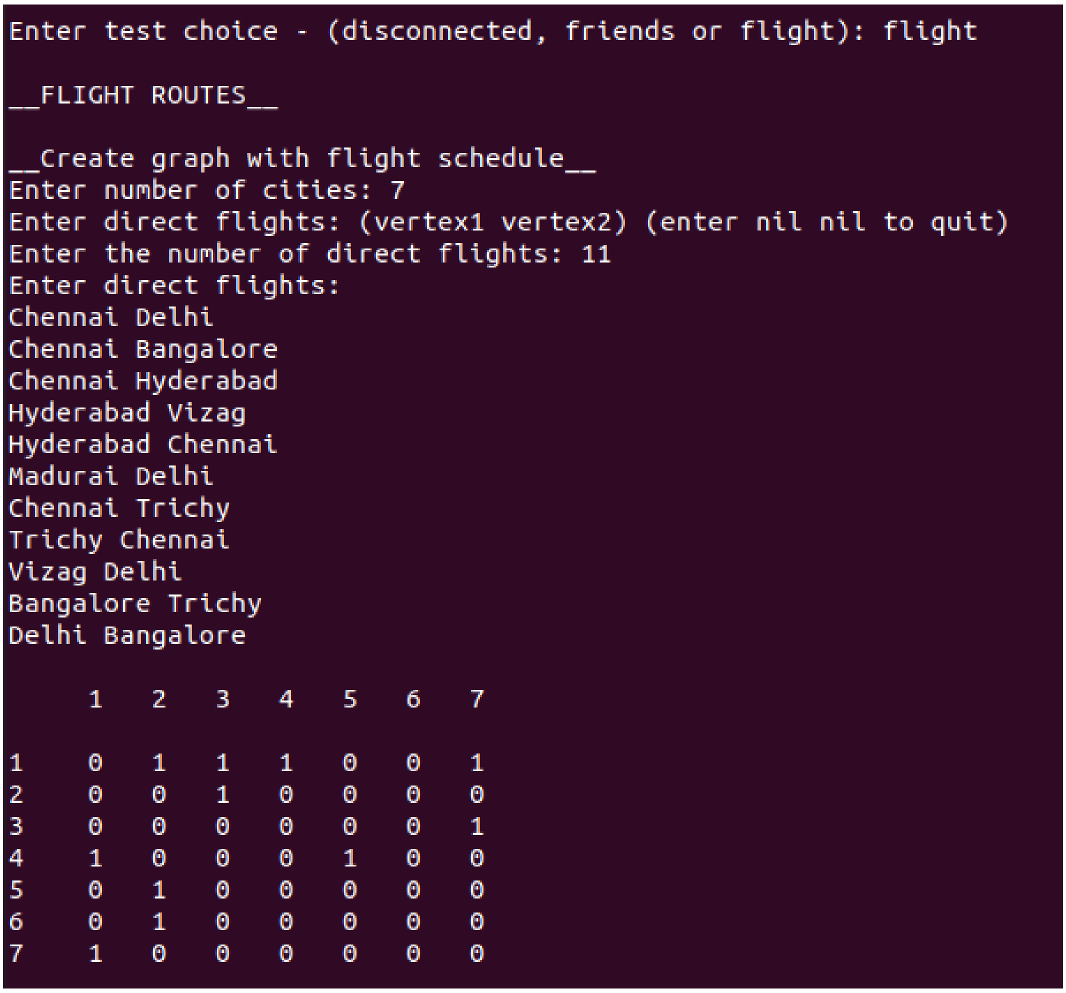
```
Enter test choice - (disconnected, friends or flight): disconnected

__TESTING A DISCONNECTED GRAPH__

__Create graph__
Enter number of vertices: 5
Enter edges: (vertex1-vertex2) (enter invalid edge to quit)
1-2
4-5
-1-0


Adjacency matrix of the disconnected graph:
     1   2   3   4   5

1    0   1   0   0   0
2    1   0   0   0   0
3    0   0   0   0   0
4    0   0   0   0   1
5    0   0   0   1   0

Enter starting vertex: 1

BFS Traversal:
1 2
3
4 5
```



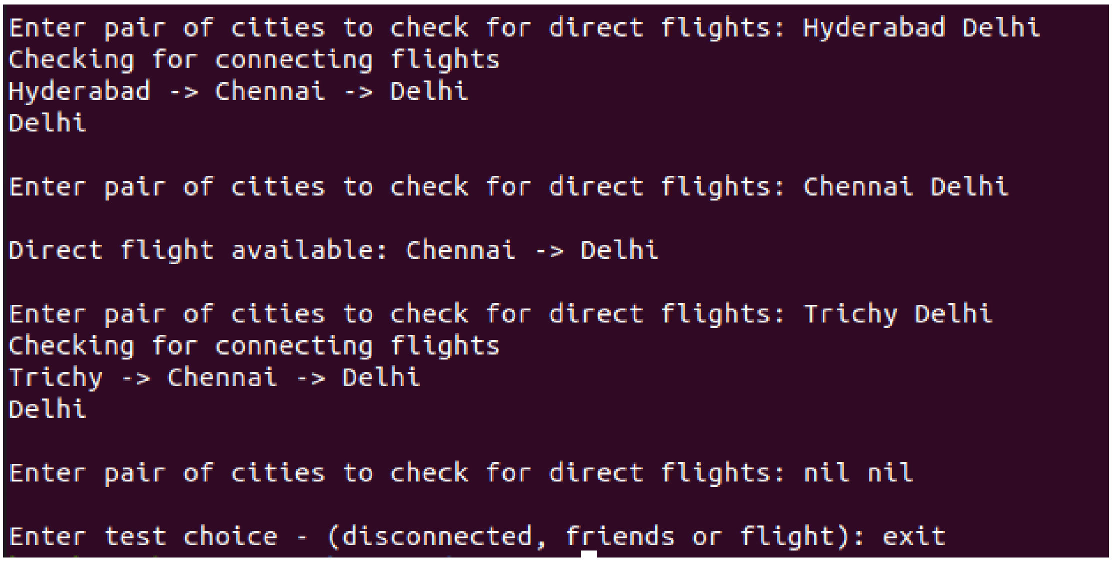
```
Enter test choice - (disconnected, friends or flight): friends

__FRIENDSHIP RELATION__

__Create graph with people as vertices and relations given by edges__
Enter number of people: 5
Enter pair of friends: (vertex1-vertex2) (enter invalid friendship to quit)
1-2
2-3
3-4
1-5
-1-0


Adjacency matrix of the friendship graph:
     1   2   3   4   5

1    0   1   0   0   1
2    1   0   1   0   0
3    0   1   0   1   0
4    0   0   1   0   0
5    1   0   0   0   0

Enter pair of people to introduce: 4 2


4 3 2
They can be introduced!
```



```
Enter test choice - (disconnected, friends or flight): friends

__FRIENDSHIP RELATION__

__Create graph with people as vertices and relations given by edges__
Enter number of people: 3
Enter pair of friends: (vertex1-vertex2) (enter invalid friendship to quit)
1-3
-1-1


Adjacency matrix of the friendship graph:
     1   2   3

1    0   0   1
2    0   0   0
3    1   0   0

Enter pair of people to introduce: 1 2


1 3
The given pair of people cannot be introduced!
```



```
Enter test choice - (disconnected, friends or flight): flight

__FLIGHT ROUTES__

__Create graph with flight schedule__
Enter number of cities: 7
Enter direct flights: (vertex1 vertex2) (enter nil nil to quit)
Enter the number of direct flights: 11
Enter direct flights:
Chennai Delhi
Chennai Bangalore
Chennai Hyderabad
Hyderabad Vizag
Hyderabad Chennai
Madurai Delhi
Chennai Trichy
Trichy Chennai
Vizag Delhi
Bangalore Trichy
Delhi Bangalore

     1   2   3   4   5   6   7

1    0   1   1   1   0   0   1
2    0   0   1   0   0   0   0
3    0   0   0   0   0   0   1
4    1   0   0   0   1   0   0
5    0   1   0   0   0   0   0
6    0   1   0   0   0   0   0
7    1   0   0   0   0   0   0
```



```
Enter pair of cities to check for direct flights: Hyderabad Delhi
Checking for connecting flights
Hyderabad -> Chennai -> Delhi
Delhi

Enter pair of cities to check for direct flights: Chennai Delhi

Direct flight available: Chennai -> Delhi

Enter pair of cities to check for direct flights: Trichy Delhi
Checking for connecting flights
Trichy -> Chennai -> Delhi
Delhi

Enter pair of cities to check for direct flights: nil nil

Enter test choice - (disconnected, friends or flight): exit
```

================================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** |  |  |  |

==================================================================

## A10: Searching and Sorting Techniques

**Date of submission: 15-12-2021**

**Question:**

1. Given an array of sorted integers with duplicate elements, write efficient C function
getfirstOccur(a[ ], size, target) that returns the index of the first occurrence of target element in the array
getlastOccur(a[ ], size, target) that returns the index of the last occurrence of target element in the array.
2. Write an efficient function in C CountOnes(a[], size) that counts the number of 1's in a sorted binary array.
3. You are given an array a, size of the array N. Follow selection sorting method and print the state of the array after each iteration has been performed.

**Code:**

### a10.h

```
//Searching and Sorting techniques

#define MAX 20

int getFirstOccur (int[], int, int);
int getLastOccur (int[], int, int);
void printArray (int[], int);
int findMinLoc (int[], int, int);
void selectionSort (int[], int);
int countOnes (int[], int);

int getFirstOccur (int a[], int size, int target) {
	int low=0, high=size-1, mid;
	while (low <= high) {
		mid = (low+high)/2;
		if (a[mid] == target && (mid==0 || a[mid-1]<target))
			return mid;
		else if (a[mid] < target)
			low = mid+1;
		else
			high = mid-1;
```

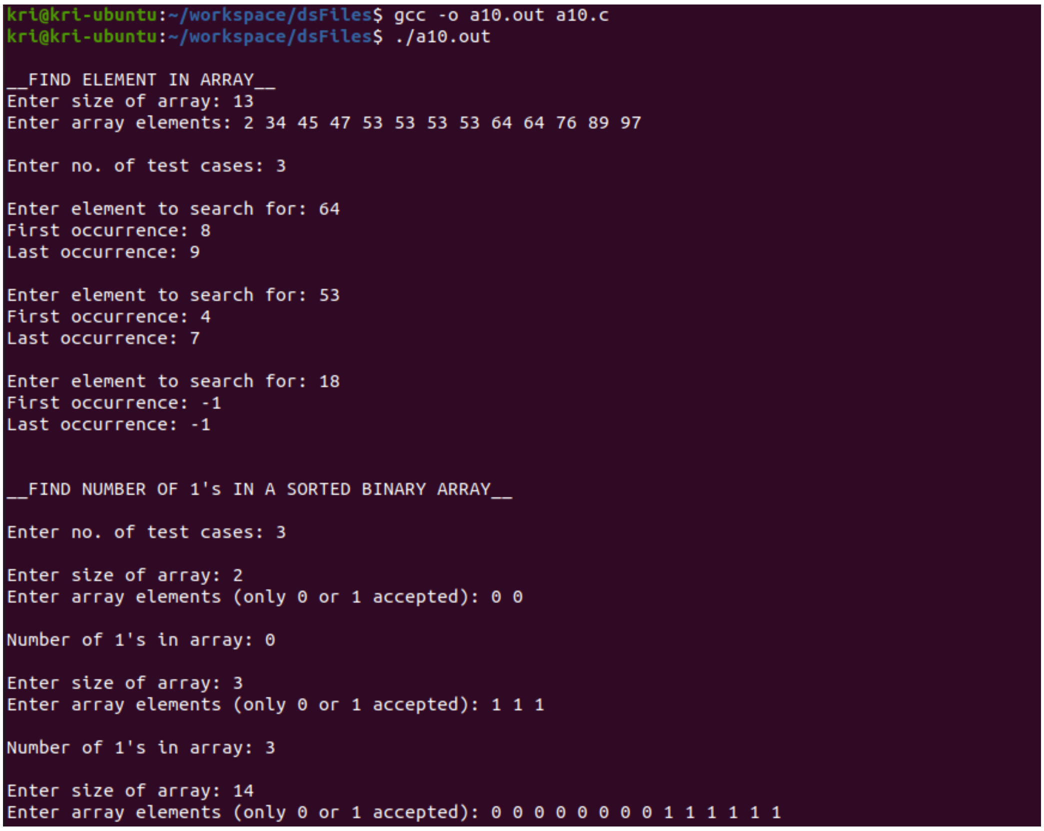
```
                }
        return -1;
        }

int getLastOccur (int a[], int size, int target) {
        int low=0, high=size-1, mid;
        while (low <= high) {
                mid = low+(high-low)/2;
                if (a[mid] == target && (mid==size-1 || a[mid+1]>target))
                        return mid;
                else if (a[mid] > target)
                        high = mid-1;
                else
                        low = mid+1;
                }
        return -1;
        }

int countOnes (int a[], int size) {
        if (a[size-1]==0)
                return 0;
        else if (a[0]==1)
                return size;
        else {
                int pos = -1, target =1;
                int low=0, high=size-1, mid;
                while (low <= high) {
                        mid = low+(high-low)/2;
                        if (a[mid] == target && (mid==0 || a[mid-1]<target))
                                pos = mid;
                        else if (a[mid] < target)
                                low = mid+1;
                        else
                                high = mid-1;
                        }
                if (pos==-1)
                                return 0;
                else
                        return size-pos;
                }
        return -1;
        }

int findMinLoc (int a[], int k, int size) {
        int j, pos;
        pos= k;
        for (j=k+1; j<size; j++)
                if (a[j] < a[pos])
                        pos= j;
        return pos;
        }
```

```
void selectionSort (int a[], int size) {
	int k, m, temp;
	for (k=0; k<size-1; k++) {
		m = findMinLoc(a, k, size);
		temp = a[k];
		a[k] = a[m];
		a[m] = temp;
		printArray(a,size);
		}
	}

void printArray (int a[], int size) {
	for (int i=0; i<size; i++)
		printf("%d ",a[i]);
	printf("\n");
	}
```

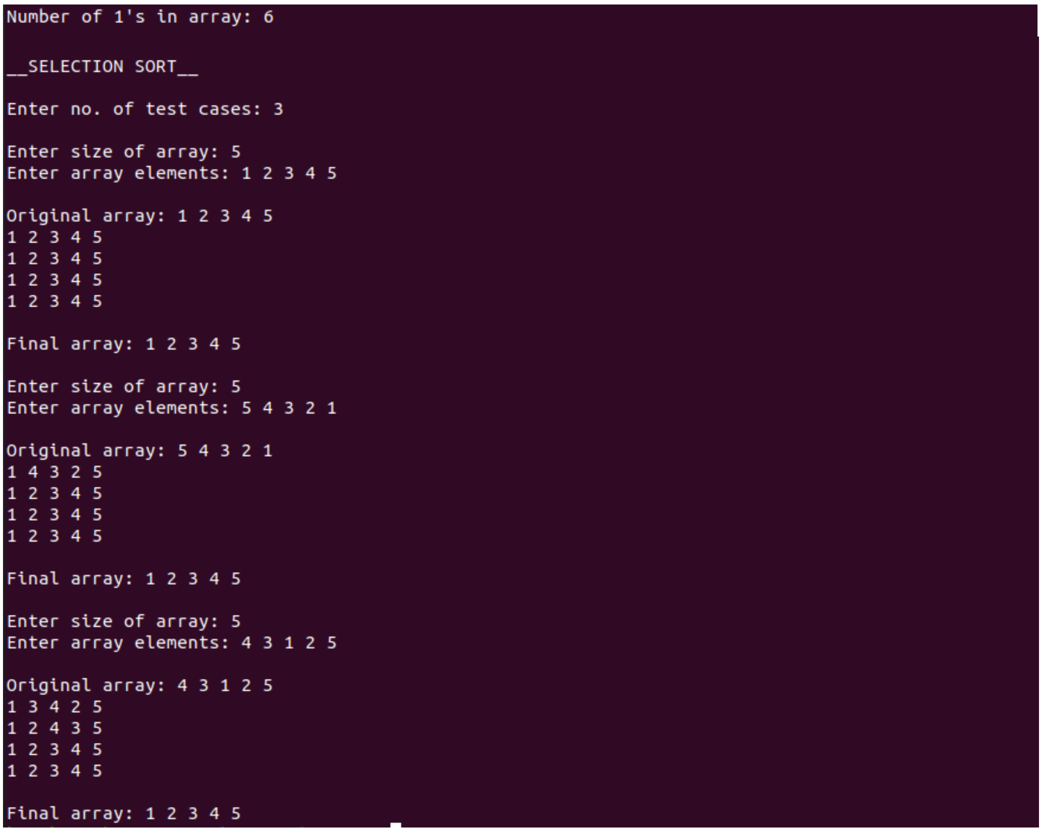
**a10.c**

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

int main() {
	int size, arr[MAX], T;

	printf("\n__FIND ELEMENT IN ARRAY__\n");

	printf("Enter size of array: ");
	scanf("%d",&size);

	printf("Enter array elements: ");
	for (int i=0; i<size; i++)
		scanf("%d",&arr[i]);

	printf("\nEnter no. of test cases: ");
	scanf("%d",&T);

	int target;
	while (T--) {
		printf("\nEnter element to search for: ");
		scanf("%d",&target);

		printf("First occurrence: %d\n",getFirstOccur(arr,size,target));
		printf("Last occurrence: %d\n",getLastOccur(arr,size,target));
		}

	printf("\n\n__FIND NUMBER OF 1's IN A SORTED BINARY ARRAY__\n");

	printf("\nEnter no. of test cases: ");
	scanf("%d",&T);

	while (T--) {
        printf("\nEnter size of array: ");
        scanf("%d",&size);

        printf("Enter array elements (only 0 or 1 accepted): ");
        for (int i=0; i<size; i++) {
                int check;
                scanf("%d",&check);
                if (!(check==0 || check==1) || (check<arr[i-1]))
                        i=-1;
                else
                        arr[i] = check;
                if (i==-1) {
                        printf("Invalid entry! Enter array again.\n");
                        }
                }

        printf("\nNumber of 1's in array: %d\n",countOnes(arr,size));
        }

printf("\n\n__SELECTION SORT__\n");

printf("\nEnter no. of test cases: ");
scanf("%d",&T);

while (T--) {
        printf("\nEnter size of array: ");
        scanf("%d",&size);

        printf("Enter array elements: ");
        for (int i=0; i<size; i++)
                scanf("%d",&arr[i]);

        printf("\nOriginal array: ");
        printArray(arr,size);

        selectionSort(arr,size);

        printf("\nFinal array: ");
        printArray(arr,size);
        }

return 0;
}
```

## Output:



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a10.out a10.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a10.out

__FIND ELEMENT IN ARRAY__
Enter size of array: 13
Enter array elements: 2 34 45 47 53 53 53 53 64 64 76 89 97

Enter no. of test cases: 3

Enter element to search for: 64
First occurrence: 8
Last occurrence: 9

Enter element to search for: 53
First occurrence: 4
Last occurrence: 7

Enter element to search for: 18
First occurrence: -1
Last occurrence: -1


__FIND NUMBER OF 1's IN A SORTED BINARY ARRAY__

Enter no. of test cases: 3

Enter size of array: 2
Enter array elements (only 0 or 1 accepted): 0 0

Number of 1's in array: 0

Enter size of array: 3
Enter array elements (only 0 or 1 accepted): 1 1 1

Number of 1's in array: 3

Enter size of array: 14
Enter array elements (only 0 or 1 accepted): 0 0 0 0 0 0 0 0 1 1 1 1 1 1
```



```
Number of 1's in array: 6

__SELECTION SORT__

Enter no. of test cases: 3

Enter size of array: 5
Enter array elements: 1 2 3 4 5

Original array: 1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

Final array: 1 2 3 4 5

Enter size of array: 5
Enter array elements: 5 4 3 2 1

Original array: 5 4 3 2 1
1 4 3 2 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

Final array: 1 2 3 4 5

Enter size of array: 5
Enter array elements: 4 3 1 2 5

Original array: 4 3 1 2 5
1 3 4 2 5
1 2 4 3 5
1 2 3 4 5
1 2 3 4 5

Final array: 1 2 3 4 5
```

======================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

==================================================================

# A11: Dijsktra’s Shortest Path Finding Technique

**Date of submission: 15-12-2021**

**Question:**

Given a graph G, implement Dijkstra’s shortest path technique to find shortest path between a given node and the rest of the nodes. Print the path too.

**Code:**

### pqHeap.h

```
//MinHeap as priority queue - ADT Structure using arrays - each node contains
key-value pair
#define EMPTY -1

struct pqMinHeap {
        unsigned capacity;
        int size;
        int* key;
        int* value;
        };
typedef struct pqMinHeap* pqHeap;

pqHeap createPQ (unsigned);
pqHeap insertPQ (pqHeap,int,int);
int deleteMin (pqHeap);
int parent (int);
int leftChild (int);
int rightChild (int);
void swap (int[],int,int);
void heapifyUp (pqHeap,int);
void heapifyDown (pqHeap,int);
int isEmpty (pqHeap);
int isFull (pqHeap);

pqHeap createPQ (unsigned capacity) {
        pqHeap heap = (pqHeap) malloc (sizeof(struct pqMinHeap));
        heap->capacity = capacity;
        heap->size = -1;
        heap->key = (int*) malloc (heap->capacity * sizeof(int));
        heap->value = (int*) malloc (heap->capacity * sizeof(int));
        return heap;
        }

pqHeap insertPQ (pqHeap heap, int key, int value) {
        //if heap is full
        if (isFull(heap))
                return heap;
        //insert key into heap as last node
        heap->key[++heap->size] = key;
        heap->value[heap->size] = value;
        //ensure heap structure
        heapifyUp(heap,heap->size);
        return heap;
        }

int deleteMin (pqHeap heap) {
        //if heap is empty
        if (isEmpty(heap))
                return EMPTY;
        //swap root node key and last node key
        swap(heap->key,0,heap->size);
        swap(heap->value,0,heap->size);
        //delete last node
        int temp = heap->value[heap->size--];
        //ensure heap structure
        heapifyDown(heap,0);
        //return minimum key
        return temp; //check - RETURN KEY OR VALUE??? FOR DIJKSTRA
        }

void heapifyUp (pqHeap heap, int i) {
        while (i>0 && (heap->key[parent(i)] > heap->key[i])) {
                //swap parent node key and current node key
                swap(heap->key,parent(i),i);
                swap(heap->value,parent(i),i);
                //update i to parent of i
                i = parent(i);
                }
        }

void heapifyDown (pqHeap heap, int i) {
        int minIndex = i;
        //left child check
        int l = leftChild(i);
        if ((l <= heap->size) && (heap->key[l] < heap->key[minIndex]))
                minIndex = l;
        //right child check
        int r = rightChild(i);
        if ((r <= heap->size) && (heap->key[r] < heap->key[minIndex]))
                minIndex = r;
        //swap minimum key with current node key
        if (i!=minIndex) {
                swap(heap->key,minIndex,i);
                swap(heap->value,minIndex,i);
                heapifyDown(heap,minIndex);
                }
        }

int isEmpty (pqHeap heap) {
        return heap->size == -1;
        }

int isFull (pqHeap heap) {
        return (heap->size!=-1 && (heap->size >= heap->capacity-1));
        }

int parent (int i) {
        return (i-1)/2;
        }

int leftChild (int i) {
        return (2*i)+1;
        }

int rightChild (int i) {
        return (2*i)+2;
        }

void swap (int heapArr[], int pos1, int pos2) {
        int temp = heapArr[pos1];
        heapArr[pos1] = heapArr[pos2];
        heapArr[pos2] = temp;
        }
```

**dijGraph.h**

```
//Graph structure for Dijkstra's algorithm

#include <stdbool.h>
#include "pqHeap.h"
#define MAX_V 20
#define INF 200

struct Graph {
        int adj[MAX_V][MAX_V];
        int size;
        };
typedef struct Graph* graph;

void initVertices(graph G, int size) {
        G->size = size;
        }
```

```
void insertEdge(graph G, int vertex1, int vertex2, int weight) {
        G->adj[vertex1][vertex2] = weight;
        G->adj[vertex2][vertex1] = weight;
        }

void printAdjMatrix (graph G) {
        int size = G->size;
        printf("\n\n     ");
        for (int i=0; i<size; i++)
                printf("%c   ",i+'A');
        printf("\n\n");
        for (int i=0; i<size; i++) {
                printf("%c    ",i+'A');
                for (int j=0; j<size; j++) {
                        int weight = G->adj[i][j];
                        if (weight>=10)
                                printf("%d  ",G->adj[i][j]);
                        else
                                printf("%d   ",G->adj[i][j]);
                        }
                printf("\n");
                }
        printf("\n");
        }
```

**dijkstra.h**

```
//Graph structure for Dijkstra's algorithm

#include <stdbool.h>
#define MAX_V 20
#define INF 200

struct Graph {
        int adj[MAX_V][MAX_V];
        int size;
        };
typedef struct Graph* graph;

int minDistance (graph G, int dist[], bool shortest[]) {
        int min = INF, min_index;

        for (int v=0; v<G->size; v++) {
                if (!shortest[v] && dist[v]<min) {
                        min = dist[v];
                        min_index = v;
                        }
                }

        return min_index;
        }
```

```
void initVertices(graph G, int size) {
        G->size = size;
        }

void insertEdge(graph G, int vertex1, int vertex2, int weight) {
        G->adj[vertex1][vertex2] = weight;
        G->adj[vertex2][vertex1] = weight;
        }

void printAdjMatrix (graph G) {
        int size = G->size;
        printf("    ");
        for (int i=1; i<=size; i++)
                printf("%d  ",i);
        printf("\n\n");
        for (int i=0; i<size; i++) {
                printf("%d   ",i+1);
                for (int j=0; j<size; j++) {
                        int weight = G->adj[i][j];
                        if (weight>=10)
                                printf("%d ",G->adj[i][j]);
                        else
                                printf("%d  ",G->adj[i][j]);
                        }
                printf("\n");
                }
        printf("\n");
        }
```

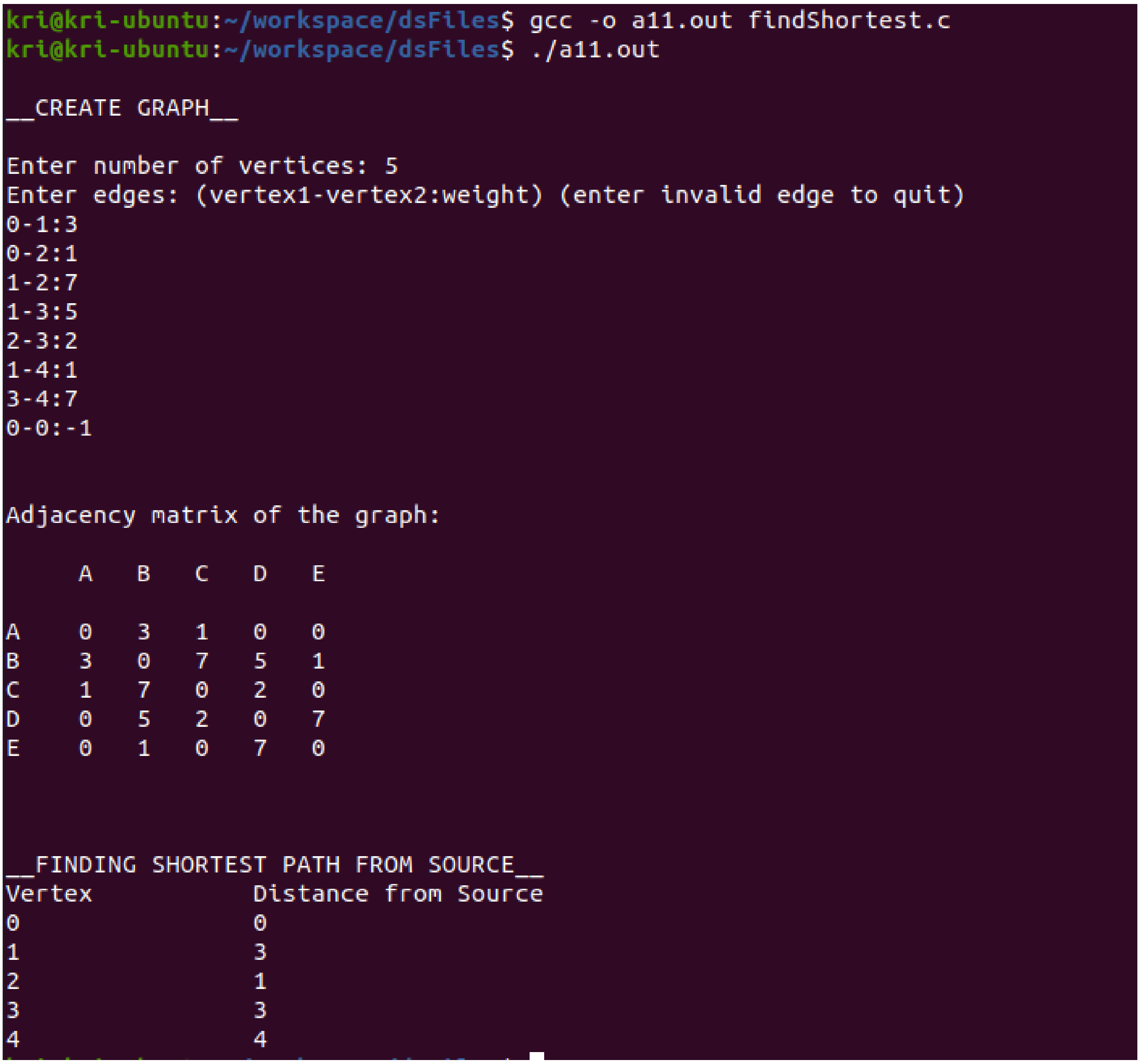
**findShortest.c**

```
//implementing dijkstra's algorithm to find the shortest path to different destinations from a single source

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

void dijkstra(graph,int);
void printSolution(int[],int);

void dijkstra (graph G, int source) {
        int dist[G->size];
        pqHeap shortest = createPQ(G->size);

        for (int i=0; i<G->size; i++)
                dist[i] = INF;

        dist[source] = 0;
        shortest = insertPQ(shortest,dist[source],source);
```

```
    while (!isEmpty(shortest)) {
        int closeV = deleteMin(shortest);

        for (int v=0; v<G->size; v++) {
            if (G->adj[closeV][v] && dist[closeV]!=INF && dist[closeV]
+G->adj[closeV][v]<dist[v]) {
                dist[v] = dist[closeV] + G->adj[closeV][v];
                shortest = insertPQ(shortest,dist[v],v);
                }
            }
        }

    printSolution(dist,G->size);
    }

int main() {
    int numV, numE, maxE;
    graph g1 = (graph) malloc (sizeof(struct Graph));

    printf("\n__CREATE GRAPH__\n\n");
    printf("Enter number of vertices: ");
    scanf("%d",&numV);
    initVertices(g1,numV);

    maxE = numV*(numV-1)/2;

    printf("Enter edges: (vertex1-vertex2:weight) (enter invalid edge to
quit)\n");
    int v1,v2,weight;
    numE=0;
    do {
        scanf("%d-%d:%d",&v1,&v2,&weight);
        if (v1>=0 && v2>=0 && weight>=0) {
            insertEdge(g1,v1,v2,weight);
            numE++;
            }
        } while(numE<=maxE && (v1>=0 && v2>=0 && weight>=0));

    printf("\n\nAdjacency matrix of the graph:");
    printAdjMatrix(g1);

    printf("\n\n__FINDING SHORTEST PATH FROM SOURCE__\n");
    int src;
    src=0;
    dijkstra(g1,src);

    return 0;
    }

void printSolution (int dist[], int numV) {
    printf("Vertex \t\t Distance from Source\n");
    for (int i=0; i<numV; i++)
```

```
            printf("%d \t\t %d\n",i,dist[i]);
    }
```

**Output:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a11.out findShortest.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a11.out

__CREATE GRAPH__

Enter number of vertices: 5
Enter edges: (vertex1-vertex2:weight) (enter invalid edge to quit)
0-1:3
0-2:1
1-2:7
1-3:5
2-3:2
1-4:1
3-4:7
0-0:-1


Adjacency matrix of the graph:

     A   B   C   D   E

A    0   3   1   0   0
B    3   0   7   5   1
C    1   7   0   2   0
D    0   5   2   0   7
E    0   1   0   7   0



__FINDING SHORTEST PATH FROM SOURCE__
Vertex           Distance from Source
0                0
1                3
2                1
3                3
4                4
```

==========================================================================

**Sri Sivasubramaniya Nadar College of Engineering**
**(An Autonomous Institution affiliated to Anna University)**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# UCS 1312 Data Structures Lab

**B.E. (CSE) III Semester**

| | | | | | |
|---|---|---|---|---|---|
| **Batch** | **:** | **2020-2024** | **Academic Year** | **:** | **2021-2022 ODD** |
| **Name** | **:** | **Krithika Swaminathan** | **Instructor** | **:** | **Dr. R. Kanchana** |
| **Roll Number** | **:** | **205001057** | | | |

==================================================================

# A12: Applications of Hash Table

**Date of submission: 15-12-2021**

**Question:**

1. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of table must be greater than or equal to total number of keys (Note that we can increase table size by copying old data if needed).

- Insert(k) – Keep probing until an empty slot is found. Once an empty slot is found, insert k.
- Search(k) – Keep probing until slot's key doesn't become equal to k or an empty slot is reached.
- Delete(k) – Delete operation is interesting. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted". Note that Insert can insert an item in a deleted slot, but search doesn't stop at a deleted slot.

2. Given an array of integers, and a number 'sum', find the number of pairs of integers in the array whose sum is equal to 'sum'. Use a hash table. Display the pairs too.

**Code:**

## hashNode.h

```
//Hash node ADT Structure

struct HashNode {
	int key;
	int value;
	};
typedef struct HashNode* hnode;
```

## hash.h

```
//Hash table ADT Structure
#define MAX_CAP 20
#include "hashNode.h"

struct HashTable {
	hnode* arr;
	unsigned capacity;
	int size;
	};
```

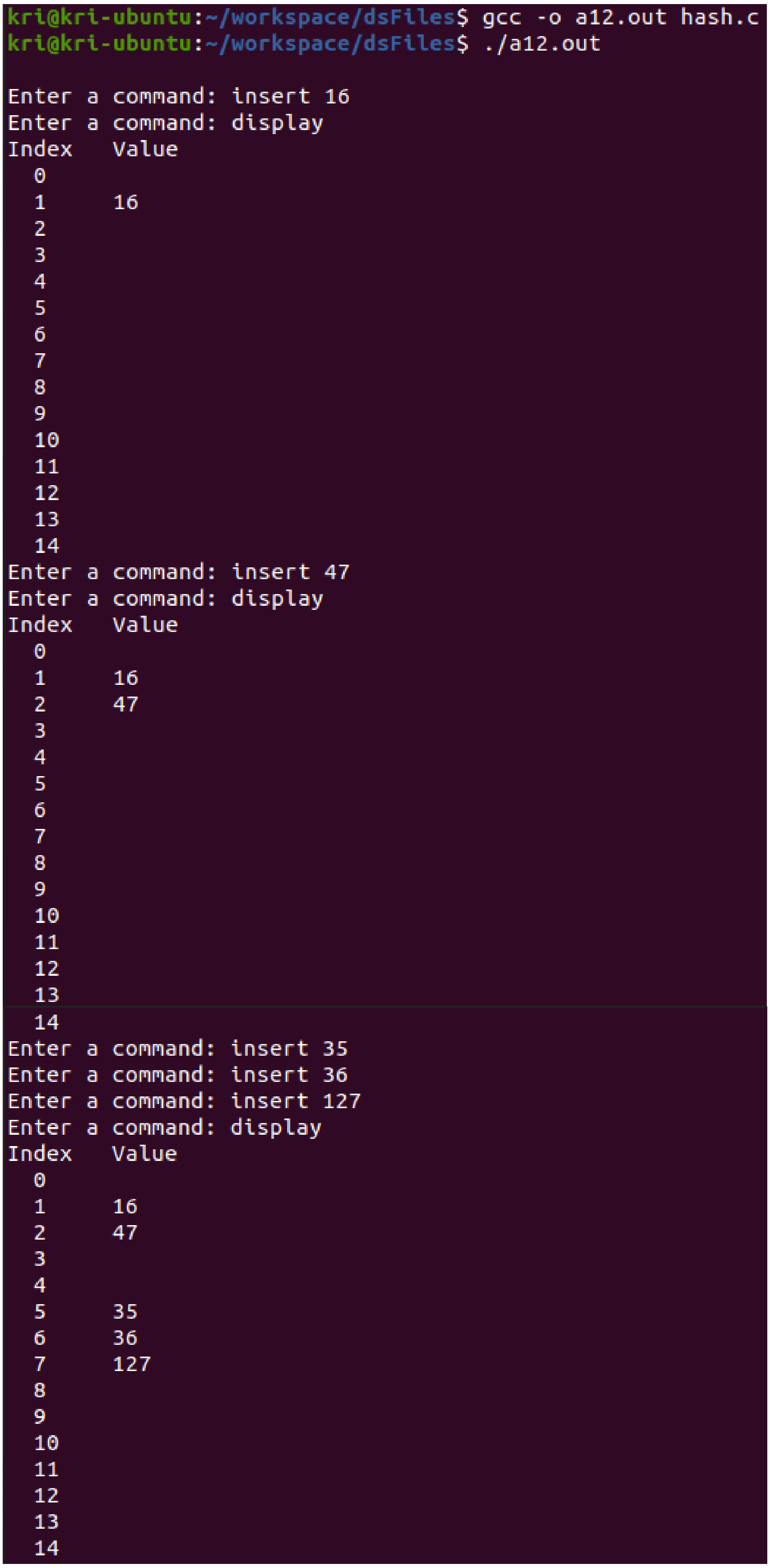
```
typedef struct HashTable* htable;

htable createTable (unsigned capacity);
int hashFunction (htable h, int key);
htable insertValue (htable h, int key, int value);
int deleteValue (htable h, int key);
int search (htable h, int key);

htable createTable (unsigned capacity) {
        htable newTable = (htable) malloc (sizeof(struct HashNode));
        newTable->size = 0;
        newTable->capacity = capacity;
        newTable->arr = (hnode*) malloc (sizeof(hnode)*capacity);
        for (int i=0; i<capacity; i++) {
                newTable->arr[i] = NULL;
                }
        return newTable;
        }

int hashFunction (htable h, int key) {
        return key%(h->capacity);
        }

htable insertValue (htable h, int key, int value) {
        if (search(h,key)!=-1)
                return insertValue(h,key+1,value);

        hnode newNode = (hnode) malloc (sizeof(struct HashNode));
        newNode->key = key;
        newNode->value = value;

        //Hash function to find index
        int hashIndex = hashFunction(h,key);

        while(h->arr[hashIndex] != NULL && h->arr[hashIndex]->key != key
&& h->arr[hashIndex]->key != -1) {
                hashIndex++;
                hashIndex %= h->capacity;
                }

        if (h->arr[hashIndex] == NULL)
                h->size++;
        h->arr[hashIndex] = newNode;

        return h;
        }

int deleteValue (htable h, int key) {
        hnode mark = (hnode) malloc (sizeof(struct HashNode));
        mark->key = -1;
        mark->value = -1;
        int hashIndex = hashFunction(h,key);
        while (h->arr[hashIndex] != NULL) {
                if(h->arr[hashIndex]->key == key) {
                        h->arr[hashIndex] = mark;
                        h->size--;
                        return 1;
                        }
                hashIndex++;
                hashIndex %= h->capacity;
                }
        return -1;
        }

int search (htable h, int key) {
        int hashIndex = hashFunction(h,key);
        int counter = 0;
        while (h->arr[hashIndex] != NULL) {
                counter = 0;
                if (counter++ > h->capacity)
                        break;
                if (h->arr[hashIndex]->key == key)
                        return h->arr[hashIndex]->value;

                hashIndex++;
                hashIndex %= h->capacity;
                }
        return -1;
        }
```

**hash.c**

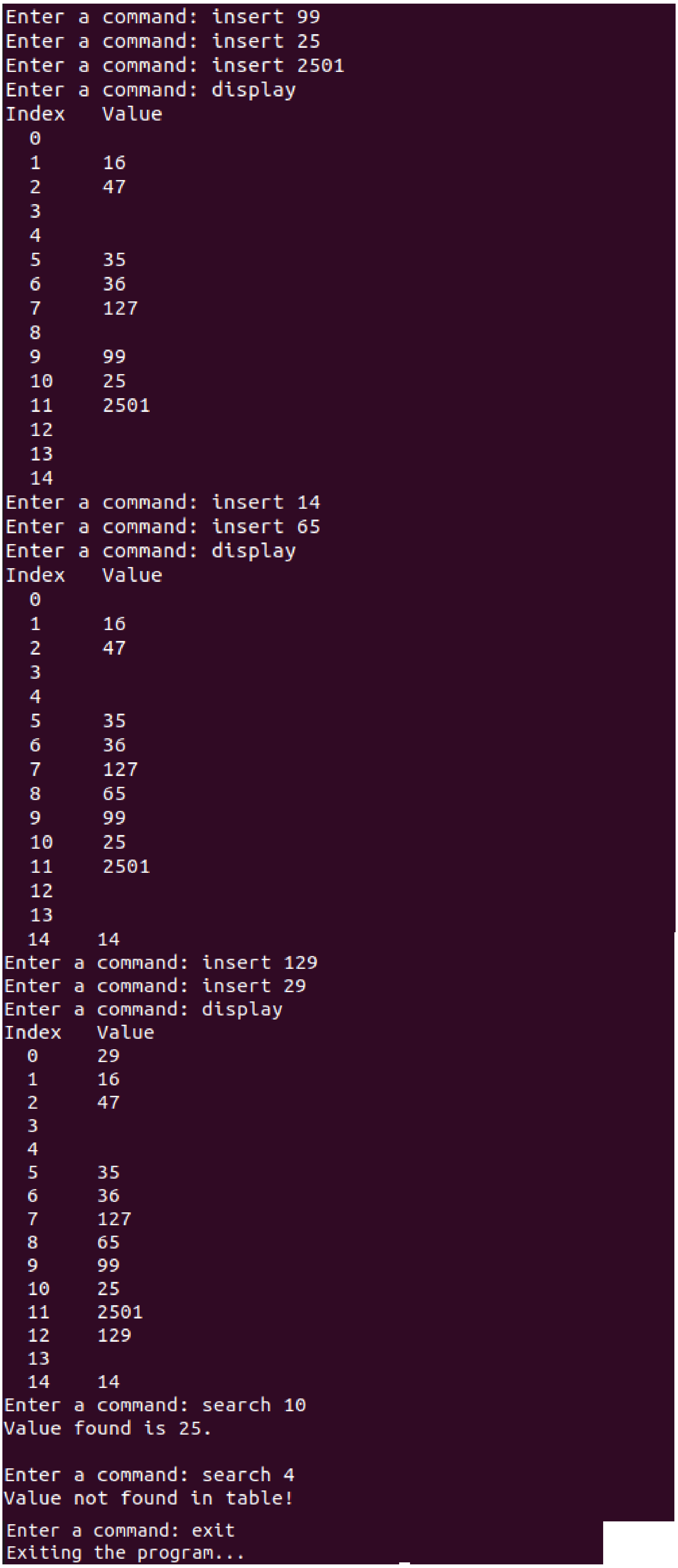
```
//implementation of hash table
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "hash.h"

void display(htable);
void sumPairs (htable,int);

int main() {
        char cmd[50]; int ch=0;
        htable hash = createTable(15);

        //enter command as input from the user
        printf("\nEnter a command: ");
        scanf("%[^\n]s",cmd);

        char* token[2];
        token[0] = strtok(cmd," ");
        token[1] = strtok(NULL," ");
```

```
do {
        if (strcmp(token[0],"insert")==0) {
                int key, value;
                value = atoi(token[1]);
                key = value % hash->capacity;
                hash = insertValue(hash,key,value);
                }
        else if (strcmp(token[0],"display")==0) {
                display(hash);
                }
        else if (strcmp(token[0],"search")==0) {
                int key = atoi(token[1]);
                int result = search(hash,key);
                if (result == -1)
                        printf("Value not found in table!\n\n");
                else
                        printf("Value found is %d.\n\n",result);
                }
        else if (strcmp(token[0],"sum")==0) {
                int sum = atoi(token[1]);
                sumPairs(hash,sum);
                }
        else if (strcmp(token[0],"Sum")==0) {
                htable h = createTable(100);
                int sum = atoi(token[1]);
                sumPairs(h,sum);
                }
        else if (strcmp(token[0],"delete")==0) {
                int key = atoi(token[1]);
                int del = deleteValue(hash,key);
                if (del==0)
                        printf("Value not found in table!\n");
                }
        else if (strcmp(token[0],"exit")==0) {
                ch=1;
                printf("Exiting the program...\n");
                exit(0);
                }
        else {
                printf("Invalid command!\n");
                exit(0);
                }

        for (int i=0; i<2; i++)
                if (token[i] != NULL)
                        memset(token[i],0,strlen(token[i]));

        printf("Enter a command: ");
        scanf(" %[^\n]s",cmd);

        token[0] = strtok(cmd," ");
        token[1] = strtok(NULL," ");
        } while (ch==0);

    return 0;
    }

void display (htable h) {
    printf("Index\tValue\n");

    for (int i = 0; i < h->capacity; i++) {
        printf("  %d\t", i);
        if (h->arr[i] == NULL)
            printf("     \n");
        else if (h->arr[i]->value == -1)
            printf("     \n");
        else
            printf("%d\n", h->arr[i]->value);
        }
    }

void sumPairs (htable h, int sum) {
    int find=0;
    for (int i=0; i<sum/2; i++) {
        if (h->arr[i] != NULL) {
            int x = sum-(h->arr[i]->value);
            if (search(h,x) != -1 && h->arr[i]->value != h->arr[x]-
>value) {
                find=1;
                //printf("%d+%d\n",h->arr[i]->value,h->arr[x]->value);
                printf("%d + %d = %d\n", h->arr[i]->value, h->arr[x]-
>value, sum);
                }
            insertValue(h,h->arr[i]->value,h->arr[i]->value);
            }
        }
    if (find==0)
        printf("No such pairs exist\n");
    }
```

**Output:**



```
kri@kri-ubuntu:~/workspace/dsFiles$ gcc -o a12.out hash.c
kri@kri-ubuntu:~/workspace/dsFiles$ ./a12.out

Enter a command: insert 16
Enter a command: display
Index   Value
  0
  1     16
  2
  3
  4
  5
  6
  7
  8
  9
  10
  11
  12
  13
  14
Enter a command: insert 47
Enter a command: display
Index   Value
  0
  1     16
  2     47
  3
  4
  5
  6
  7
  8
  9
  10
  11
  12
  13
  14
Enter a command: insert 35
Enter a command: insert 36
Enter a command: insert 127
Enter a command: display
Index   Value
  0
  1     16
  2     47
  3
  4
  5     35
  6     36
  7     127
  8
  9
  10
  11
  12
  13
  14
```



```
Enter a command: insert 99
Enter a command: insert 25
Enter a command: insert 2501
Enter a command: display
Index   Value
  0
  1     16
  2     47
  3
  4
  5     35
  6     36
  7     127
  8
  9     99
  10    25
  11    2501
  12
  13
  14
Enter a command: insert 14
Enter a command: insert 65
Enter a command: display
Index   Value
  0
  1     16
  2     47
  3
  4
  5     35
  6     36
  7     127
  8     65
  9     99
  10    25
  11    2501
  12
  13
  14    14
Enter a command: insert 129
Enter a command: insert 29
Enter a command: display
Index   Value
  0     29
  1     16
  2     47
  3
  4
  5     35
  6     36
  7     127
  8     65
  9     99
  10    25
  11    2501
  12    129
  13
  14    14
Enter a command: search 10
Value found is 25.

Enter a command: search 4
Value not found in table!
Enter a command: exit
Exiting the program...
```

==================================================================================