RAMAIAH
Institute of Technology

Department of Computer Science & Engineering
(Artificial Intelligence & Machine Learning)
and
Computer Science & Engineering
(Cyber Security)

Data Structures Laboratory

(CIL36/CYL36)

Lab Manual

(AY: 2023-24)

Prepared by:
Dr. Tojo Mathew

Table of Contents

Program 1 Array operations

Design, Develop and Implement a menu driven Program in C for the following array operations.

a) Creating an array of N Integer Elements
b) Display of array Elements with Suitable Headings
c) Inserting an Element (ELEM) at a given valid Position (POS)
d) Deleting an Element at a given valid Position(POS)
e) Exit.

Support the program with functions for each of the above operations.

Sample solution:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
int arr[MAX];
int pos, elem, n;

// Function declarations
void create();
void display();
void insert();
void delete();

int main()
{
   int choice;
   int pos = 1;
   while (1)
   {
      printf("\n1.Create an array ");
      printf("\n2.Display the array ");
      printf("\n3.Insert an element ");
      printf("\n4.Delete an element ");
      printf("\n5.Exit ");
      printf("\nEnter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         create();
         break;
      case 2:
         display();
         break;
```

```
        case 3:
           insert();
           break;
        case 4:
           delete ();
           break;
        case 5:
           exit(1);
        default:
           printf("Wrong choice ");
        }
    }
}


// Function definitions
void create()
{
    int i;
    printf("Enter the size of the array : ");
    scanf("%d", &n);
    printf("Enter the elements of the array : ");
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
}

void display()
{
    int i;
    if (n == 0)
    {
        printf("Array is empty ");
        return;
    }
    printf("Array elements are : ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf(" ");
}

void insert()
{
    int i;
    if (n == MAX)
    {
        printf("Array is full ");
        return;
    }
    printf("Enter the position for the new element : ");
    scanf("%d", &pos);
    printf("Enter the element to be inserted : ");
    scanf("%d", &elem);
    // Shifting all elements from index 'pos' by 1 position
    // towards the right to create a vacant position at 'pos.
    for (i = n - 1; i >= pos; i–)
        arr[i + 1] = arr[i];
    // Inserting new element at index 'pos'
    arr[pos] = elem;
    n = n + 1; // Incrementing total element count
}

void delete()
{
    int i;
    printf("Enter the position of the element to be deleted : ");
    scanf("%d", &pos);
    elem = arr[pos];
    printf("Element deleted is : %d ", elem);
    // Shifting all elements after index 'pos' by 1 position
    // towards left, overwriting element at pos by next one.
    for (i = pos; i <= n - 1; i++)
        arr[i] = arr[i + 1];
    n = n - 1; // Decrementing total element count
}
```

OUTPUT:

```
1.Create an array
2.Display the array
3.Insert an element
4.Delete an element
5.Exit
Enter your choice : 1
Enter the size of the array : 8
Enter the elements of the array : 1 2 3 4 5 6 7 8

1.Create an array
2.Display the array
```

```
3.Insert an element
4.Delete an element
5.Exit
Enter your choice : 2
Array elements are : 1 2 3 4 5 6 7 8
1.Create an array
2.Display the array
3.Insert an element
4.Delete an element
5.Exit
Enter your choice : 3
Enter the position for the new element : 3
Enter the element to be inserted : 99

1.Create an array
2.Display the array
3.Insert an element
4.Delete an element
5.Exit
Enter your choice : 2
Array elements are : 1 2 3 99 4 5 6 7 8
1.Create an array
2.Display the array
3.Insert an element
4.Delete an element
5.Exit
Enter your choice : 4
Enter the position of the element to be deleted : 7
Element deleted is : 7
1.Create an array
2.Display the array
3.Insert an element
4.Delete an element
5.Exit
Enter your choice : 2
Array elements are : 1 2 3 99 4 5 6 8
1.Create an array
2.Display the array
3.Insert an element
4.Delete an element
5.Exit
Enter your choice : 5
```

Program 2 Employee Structure

Define an EMPLOYEE structure with members Emp_name, Emp-id, Dept-name and Salary. Read and display data of N employees. Employees may belong to different departments. Write a function to find the total salary of employees of a specified department. Use the concept of pointer to structure and allocate the memory dynamically to EMPLOYEE instances.

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

#define MAX_NAMELEN 50
#define MAX_DEPTLEN 50

typedef struct employee
{
   char Emp_name[MAX_NAMELEN];
   int Emp_id;
   char Dept_name[MAX_DEPTLEN];
   float Salary;
} EMPLOYEE;

// Function to find the total salary of employees of a specified department
float total_salary_by_dept(EMPLOYEE *emp_array, int n, char *dept_name)
{
   float total_salary = 0;
   for (int i = 0; i < n; i++)
   {
      if (strcmp(emp_array[i].Dept_name, dept_name) == 0)
      {
         total_salary += emp_array[i].Salary;
      }
   }
   return total_salary;
}

int main()
{
   int n, i;
   char dept_name[MAX_DEPTLEN];
   EMPLOYEE *emp_array; // Pointer to structure

   printf("Enter the number of employees: ");
   scanf("%d", &n);

   emp_array = (EMPLOYEE *)malloc(n * sizeof(EMPLOYEE));
```

```
    for (i = 0; i < n; i++)
    {
        printf("Enter the name of employee %d: ", i + 1);
        scanf("%s", emp_array[i].Emp_name);
        printf("Enter the id of employee %d: ", i + 1);
        scanf("%d", &emp_array[i].Emp_id);
        printf("Enter the department of employee %d: ", i + 1);
        scanf("%s", emp_array[i].Dept_name);
        printf("Enter the salary of employee %d: ", i + 1);
        scanf("%f", &emp_array[i].Salary);
    }

    printf("Enter the department name: ");
    scanf("%s", dept_name);
    float total_salary = total_salary_by_dept(emp_array, n, dept_name);

    printf("Total salary of employees in %s department is %.2f", dept_name, total_salary);
    free(emp_array);

    return 0;
}
```

OUTPUT

```
Enter no of employee
3
Enter the employee details for 3 employees:

Enter the name of 1 employee : tabraiz
Enter employee 1 id : 10
Enter dept of employee 1 : cse
Enter employee 1 salary : 70000
Enter the name of 2 employee : kshitij
Enter employee 2 id : 20
Enter dept of employee 2 : cse
Enter employee 2 salary : 80000
Enter the name of 3 employee : jeet
Enter employee 3 id : 30
Enter dept of employee 3 : ise
Enter employee 3 salary : 60000
The employee details are :
Employee 1 name :  tabraiz
Employee 1 id : 10
Employee 1 dept : cse
Employee 1 salary : 70000.000000
Employee 2 name :  kshitij
Employee 2 id : 20
Employee 2 dept : cse
Employee 2 salary : 80000.000000
Employee 3 name :  jeet
Employee 3 id : 30
Employee 3 dept : ise
Employee 3 salary : 60000.000000
Enter the department name
cse
the total salary of the cse department is 150000.000000
```

Program 3 Stack Implementation

STACK of Integers (Array Implementation of Stack with maximum size MAX)

a) Push an Element onto Stack.
b) Pop an Element from Stack.
c) Demonstrate how Stack can be used to check Palindrome.
d) Demonstrate Overflow and Underflow situations on Stack
e) Display the status of Stack
f) Exit

Support the program with appropriate functions for each of the above operations

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

int stack[MAX];
int top = -1;

void push(int x);
int pop();
void display();
// Check if a string is palindrome
void check_palindrome();
// Check if top & bottom half of stack is mirror image of each other
void check_palindrome2();


int main()
{
   int choice, x;
   while (1)
   {
      printf("\n1.Push ");
      printf("\n2.Pop ");
      printf("\n3.Display ");
      printf("\n4.Quit ");
      printf("\n5.Check Palindrome ");
      printf("\nEnter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("Enter the element to be pushed : ");
         scanf("%d", &x);
         push(x);
```

```
            break;
        case 2:
            x = pop();
            printf("Popped element is : %d", x);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(1);
        case 5:
            check_palindrome();
            check_palindrome2();
            break;
        default:
            printf("Invalid choice ");
        }
    }
}


void push(int x)
{
    if (top == MAX - 1)
        printf("Stack Overflow ");
    else
    {
        top = top + 1;
        stack[top] = x;
    }
}

int pop()
{
    int x;
    if (top == -1)
        printf("Stack Underflow ");
    else
    {
        x = stack[top];
        top = top - 1;
        return x;
    }
}

void display()
{
    int i;
    if (top == -1)
        printf("Stack is empty ");
    else
    {
        printf("Stack elements are : ");
        for (i = top; i >= 0; i--)
            printf("%d ", stack[i]);
    }
}

// Check if a string is palindrome
void check_palindrome()
{
    int i, j, flag = 0;
    char str[20];
    printf("Resetting stack\n");
    top = -1;
    printf("Enter the string : ");
    scanf("%s", str);
    for (i = 0; str[i] != '\0'; i++)
        push(str[i]);
    for (j = 0; str[j] != '\0'; j++)
    {
        if (str[j] != pop()) // str[j] is first element of string and pop() is last element of string
        {
            flag = 1; // if first and last elements are not same then flag is set to 1
            break;
        }
    }
    if (flag == 1)
        printf("String is not a palindrome\n");
    else
        printf("String is a palindrome\n");
}

// Check if top & bottom half of stack are mirror image of each other
void check_palindrome2()
{
 int floor=0,ceil=top,flag=0;
 while(floor<ceil)
 {
  if(stack[floor]!=stack[ceil])
  {
   flag=1;
   break;
  }
   floor++;
   ceil--;
 }
 if(flag==1)
  printf("Stack is not  a palindrome\n");
  else
  printf("Stack is a palindrome\n");
}
```

OUTPUT

```
1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 1
Enter the element to be pushed : 5

1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 1
Enter the element to be pushed : 6

1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 1
Enter the element to be pushed : 7

1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 3
Stack elements are : 7 6 5
1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 2
Popped element is : 7
```

```
1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 3
Stack elements are : 6 5
1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 5
Resetting stack
Enter the string : madam
String is a palindrome
Stack is a palindrome

1.Push
2.Pop
3.Display
4.Quit
5.Check Palindrome
Enter your choice : 5
Resetting stack
Enter the string : hello
String is not a palindrome
Stack is not  a palindrome
```

Program 4 Infix to Postfix Conversion

Write a C program to convert and print a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and binary operators + - * /. Apply the concept of stack data structure to solve this problem

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top=-1;
void push(char x);
char pop();
int priority(char x);
int main()
{
    char exp[100],*e,x;
    printf("Enter the infix expression\n");
    scanf("%s",exp);
    e=exp;
    while(*e!='\0')
    {
        if(isalnum(*e))
            printf("%c",*e);
        else if(*e=='(')
             push(*e);
        else if(*e==')')
        {
            while((x=pop())!='(')
            printf("%c",x);
           // push(*e);
        }
        else
        {
            while(priority(stack[top])>=priority(*e))
            printf("%c",pop());
            push(*e);
        }
        e++;
    }
    while(top!=-1)
      printf("%c",pop());
     printf("\n");
    return 0;
}
void push(char x)
{
   stack[++top]=x;
}
char pop()
{
   if(top==-1)
    return -1;
   else
    return stack[top--];
}
int priority(char x)
{
   if(x=='(')
   return 0;
   if(x=='+'||x=='-')
   return 1;
   if(x=='*'||x=='/')
   return 2;
   if(x=='$'||x=='^')
   return 3;
return 0;
}
```

OUTPUT

```
Enter the infix expression
A+B/C*(D-E)
ABC/DE-*+

Enter the infix expression
A+B*C
ABC*+
```

Program 5 Evaluation of Postfix Expression

Write a C program to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary operators. The operators are + - * and /.

```
#include <stdio.h>

// Minimalistic stack for the use in infix
// to postfix conversion
int stack[20];
int top = -1;

void push(int x)
{
    stack[++top] = x;
}

int pop()
{
    return stack[top--];
}

int main()
{
    char exp[20];
    char *e;
    int n1, n2, n3, num;
    printf("Enter the expression :: ");
    scanf("%s", exp);
    e = exp;
    while (*e != '\0')
    {
        if (isdigit(*e))
        {
            num = *e - 48; // converting char to int
            push(num);
        }
        else
        {
            n1 = pop();
            n2 = pop();
            switch (*e)
            {
            case '+':
            {
                n3 = n1 + n2;
                break;
            }
            case '-':
            {
                n3 = n2 - n1;
                break;
            }
            case '*':
            {
                n3 = n1 * n2;
                break;
            }
            case '/':
            {
                n3 = n2 / n1;
                break;
            }
            }
            push(n3);
        }
        e++;
    }
    printf("\nThe result of expression %s  =  %d\n\n", exp, pop());
    return 0;
}
```

OUTPUT

```
Enter the postfix expression
53+62/*35*+
Thes result of 53+62/*35*+ = 39
```

Program 6 Recursion applications

Write recursive functions for the following and demonstrate their use.

a) Binary Search
b) Tower of Hanoi problem

6a. Binary Search

```
// C program to implement recursive Binary Search
#include <stdio.h>

// Function declaration/prototype
int binarySearch(int arr[], int low, int high, int key);

// Driver code
int main(void)
{
   int n, x, i;
   printf("Enter the number of elements in the array: ");
   scanf("%d", &n);
   int arr[n];
   printf("Enter %d elements in the array: \n", n);
   for (i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }
   printf("Enter the element to be searched: ");
   scanf("%d", &x);
   int result = binarySearch(arr, 0, n - 1, x);
   (result == -1) ? printf("Element is not present in array")
               : printf("Element is present at index %d", result);
   return 0;
}

/* A recursive binary search function. It returns location of key
in given array arr[low,.,.,.,.,high] if present, otherwise  -1  */
int binarySearch(int arr[], int low, int high, int key)
{
      if ( low<=high) {
         int mid = (low + high) / 2;

         // If the element is present at the middle itself
         if (arr[mid] == key)
            return mid;
            // If element is smaller than mid, then
            // it can only be present in left subarray
            if (arr[mid] > key)
                return binarySearch(arr, low, mid - 1, key);

            // Else the element can only be present
            // in right subarray
            return binarySearch(arr, mid + 1, high, key);
        }

        // We reach here when element is not
        // present in array
        return -1;
}
```

OUTPUT:

```
Enter the number of elements in the array: 5
Enter 5 elements in the array:
1 4 7 9 12
Enter the element to be searched: 9
Element is present at index 3


Enter the number of elements in the array: 5
Enter 5 elements in the array:
1 4 7 9 12
Enter the element to be searched: 15
Element is not present in array
```

6b. Tower of Hanoi problem

```
#include <stdio.h>
void ToH(int n, char source,  char spare, char dest);
static int step=0;
int main()
{
   int n;
   printf("\n Enter the number of rings: ");
   scanf("%d", &n);
   // Move n rings from A to C with B as auxiliary
   ToH(n,'A', 'B', 'C');
   return 0;
}
void ToH(int n, char A,  char B, char C)
```

```
{
   if (n==1)
      printf("\n Step %d: Move %d from %c to %c",++step,n, A,C);
   else
   {
      ToH(n-1,A,C,B);
      ToH(1,A,B,C);
      ToH(n-1,B,A, C);
   }
}
```

OUTPUT

```
Enter the number of rings: 4
 Step 1: Move 1 from A to B
 Step 2: Move 1 from A to C
 Step 3: Move 1 from B to C
 Step 4: Move 1 from A to B
 Step 5: Move 1 from C to A
 Step 6: Move 1 from C to B
 Step 7: Move 1 from A to B
 Step 8: Move 1 from A to C
 Step 9: Move 1 from B to C
 Step 10: Move 1 from B to A
 Step 11: Move 1 from C to A
 Step 12: Move 1 from B to C
 Step 13: Move 1 from A to B
 Step 14: Move 1 from A to C
 Step 15: Move 1 from B to C
```

Program 7 Call holding in call centre

A Call center phone system has to hold the phone calls from customers and provide service based on the arrival time of the calls. Write a C program to simulate this system using appropriate data structure. Program should have options to add and remove the phone calls in appropriate order for their service.

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

#define MAX 3

int items[MAX];
int front = -1, rear = -1; // Initial empty state

// Queue operations
void insert(int value);
void delete();

// Supporting function for queue visualization
void display();


int main()
{
   int ch;
   int callid;
   while(1)
   {
      printf("\nEnter appropriate choice\n1.add call\n2.remove call\n3.display call
list\n4.exit\n");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:
            printf("Enter caller id to add\n");
            scanf("%d",&callid);
            insert(callid);
            break;
         case 2: delete();
              break;
         case 3: display();
              break;
            case 4: exit(0);
            default : printf("invalid choice\n");
        }
    }
}
void insert(int value){
    if(rear == MAX-1)
        printf("\nQUEUE OVERFLOW");
    else {
        if(front == -1)
            front = 0; // Going to insert first
        rear = rear+1;
        items[rear] = value;
        printf("\nCallerId added -> %d", value);
    }
}

void delete(){
    if(front == -1)
        printf("\nCALL QUEUE UNDERFLOW!!");
    else{
        printf("\nCallerId Deleted : %d", items[front]);
        front++;
        // Reset queue if last element deleted
        if(front > rear)
            front = rear = -1;
      }
}

void display(){
    if(rear == -1)
        printf("\nCall Queue is Empty!!!");
    else{
        int i;
        printf("\nCalls held in queue are:\n");
        for(i=front; i<=rear; i++)
            printf("%d  ",items[i]);
    }
}
```

OUTPUT:

```
Enter appropriate choice
1.add call
```

```
2.remove call
3.display call list
4.exit
1
Enter caller id to add
2345

CallerId added -> 2345
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
1
Enter caller id to add
7865

CallerId added -> 7865
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
1
Enter caller id to add
7777

CallerId added -> 7777
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
3

Calls held in queue are:
2345  7865  7777
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
1
Enter caller id to add
9999

QUEUE OVERFLOW
Enter appropriate choice
```

```
1.add call
2.remove call
3.display call list
4.exit
2

CallerId Deleted : 2345
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
3

Calls held in queue are:
7865  7777
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
2

CallerId Deleted : 7865
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
3

Calls held in queue are:
7777
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
2

CallerId Deleted : 7777
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
3
```

```
Call Queue is Empty!!!
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
2

CALL QUEUE UNDERFLOW!!
Enter appropriate choice
1.add call
2.remove call
3.display call list
4.exit
```

Program 8 Circular Queue using Structure

Write a C program to simulate the working of a Circular Queue of integers. Represent a circular queue element as a structure and use an array of structures as your implementation method. Start and end of the circular queue must be identified by an empty array element.

Solution 1: Using a simple structure

```
#include <stdio.h>
#define MAX_SIZE 3

struct CircularQueue
{
    int data;       // data to be stored in the queue
};

// Array of structures
struct CircularQueue queue[MAX_SIZE];
int front = -1, rear = -1;

void insert(int value);
int delete();
void display();

int main()
{
    int choice, value;
    while (1)
    {
        printf("1. Insert element in the queue\n");
        printf("2. Delete element from the queue\n");
        printf("3. Display elements in the queue\n");
        printf("4. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d", &value);
            insert(value);
            break;
        case 2:
            value = delete();
            if (value != -1)
            {
                printf("The deleted element is: %d\n", value);
            }
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}

void insert(int value)
{
    if ((front == 0 && rear == MAX_SIZE - 1) || (rear + 1 == front))
    {
        printf("Circular Queue is full\n");
        return;
    }
    if (front == -1)
    {
        front = rear = 0;
    }
    else if (rear == MAX_SIZE - 1)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    queue[rear].data = value;
}

// Function to remove an element from the queue
int delete()
{
    int value;
    if (front == -1) // queue is empty
    {
        printf("QUEUE UNDERFLOW\n");
        return -1;
    }
    value = queue[front].data;
    if (front == rear) // only one element in the queue
    {
        front = rear = -1;
    }
    else if (front == MAX_SIZE - 1) // front point to last of array
    {
        front = 0;
    }
    else // more than one element in the queue
    {
        front++;
    }
    return value;
}


void display()
{
    int i;
    if(front == -1) printf(" \n Empty Queue\n");
    else
    {
        printf("\n Front -> %d ",front);
        printf("\n Items -> ");
        for( i = front;  i!=rear;  i=(i+1)%MAX_SIZE) {
            printf("%d  ",queue[i].data);
        }
        printf("%d ",queue[i].data);  // Print last element of queue where front = rear
        printf("\n Rear -> %d \n",rear);
    }
}
```

OUTPUT

```
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 1
Enter the value to be inserted: 45
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 1
Enter the value to be inserted: 67
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
```

```
Enter your choice: 3

 Front -> 0
 Items -> 45  67
 Rear -> 1
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 1
Enter the value to be inserted: 78
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 1
Enter the value to be inserted: 99
Circular Queue is full
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 3

 Front -> 0
 Items -> 45  67  78
 Rear -> 2
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 2
The deleted element is: 45
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 3

 Front -> 1
 Items -> 67  78
 Rear -> 2
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 1
Enter the value to be inserted: 99
```

```
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
Enter your choice: 3

 Front -> 1
 Items -> 67  78  99
 Rear -> 0
1. Insert element in the queue
2. Delete element from the queue
3. Display elements in the queue
4. Quit
```

Solution 2 :Circular Queue of student records

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 3

typedef struct QueItem
{
   char name[20];
   int usn;
   float cgpa;
}QueItem;

// Array of structures
struct QueItem queue[MAX_SIZE];
int front = -1, rear = -1;

void insert(QueItem item);
int delete();
void display();

int main()
{
   int choice, value;
   while (1)
   {
      printf("1. Insert element in the queue\n");
      printf("2. Delete element from the queue\n");
      printf("3. Display elements in the queue\n");
      printf("4. Quit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
            {
            QueItem newItem;
            printf("Enter name:");
            scanf("%s", newItem.name);
            printf("Enter usn:");
            scanf("%d", &newItem.usn);
            printf("Enter cgpa:");
            scanf("%f", &newItem.cgpa);
            insert(newItem);
            break;
            }
        case 2:
            {
            QueItem item;
            value = delete(&item); //call-by-ref
            if (value != -1)
            {
                printf("The deleted element is: %s\n", item.name);
            }
            break;
            }
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}

void insert(QueItem value)
{
    if ((front == 0 && rear == MAX_SIZE - 1) \
        || (rear + 1 == front))
    {
        printf("Circular Queue is full\n");
        return;
    }
    if (front == -1)
    {
        front = rear = 0;
    }
    else
    {
        rear = (rear+1)%MAX_SIZE;
    }
    queue[rear]= value;
}

// Function to remove an element from the queue
int delete(QueItem *rItem)
{
    QueItem value;
    if (front == -1) // queue is empty
    {
        printf("QUEUE UNDERFLOW\n");
        return -1;
    }
    *rItem = queue[front];
    if (front == rear) // only one element in the queue
    {
        front = rear = -1;
    }
    else
    {
        front = (front+1)%MAX_SIZE;
    }
    return 0;
}

void display()
{
    int i;
    if(front == -1) printf(" \n Empty Queue\n");
    else
    {
        printf("\nFront -> %d\n",front);
        for( i = front;  i!=rear;  i=(i+1)%MAX_SIZE) {
            printf("Item:%s %d %.2f\n",queue[i].name,queue[i].usn,\
                   queue[i].usn);
        }
        printf("Item:%s %d %.2f",queue[i].name,queue[i].usn,\
               queue[i].usn);  // Print last element
        printf("\nRear -> %d \n",rear);
    }
}
```

Program 9 Linked list for sorted names

Write a program to create a singly linked list that maintains a list of names in alphabetical order. Implement the following operations on the list.

a. Insert a new name
b. Delete a specified name

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

typedef struct node
{
   char name[50];
   struct node *next;
} Node;

Node *head = NULL; // head of the list

void insert(char *name)
{
   Node *newNode = (Node *)malloc(sizeof(Node));
   strcpy(newNode->name, name); // copy name to newNode->name
   newNode->next = NULL;
   // check if the list is empty or the new name is smaller than the
   // first name
   if (head == NULL || strcmp(head->name, name) >= 0)
   {
      newNode->next = head; // insert the new node at the beginning
      head = newNode;       // update the head
   }
   else
   {
      Node *current = head; // current is the node before the insertion point
      // finding the insertion point using below loop
      while (current->next != NULL && strcmp(current->next->name, name) < 0)
                 current = current->next;          // move to the next node
      newNode->next = current->next;               // insert the new node after current
      current->next = newNode;                     // update the next pointer of current
   }
}

void delete(char *name)
{
   Node *temp = head, *prev;
   // check if the first node is the one to be deleted
```

```
    if (temp != NULL && strcmp(temp->name, name) == 0)
    {
        head = temp->next;
        free(temp);
        return;
    }
    // find the node to be deleted
    while (temp != NULL && strcmp(temp->name, name) != 0)
    {
        prev = temp;       // keep track of the previous node
        temp = temp->next; // move to the next node
    }
    if (temp == NULL) // check if the node is not found
    {
            printf("Given name not found!");
            return;
    }
    prev->next = temp->next; // update the next pointer of the previous node
    free(temp);            // free the memory
}


void display()
{
    Node *ptr = head;
    printf("\nNames in the List: ");
    while (ptr != NULL)
    {
        printf("%s ", ptr->name);
        ptr = ptr->next;
    }
}

int main()
{
    int choice;
    char name[50];
    while (1)
    {
        printf("\n1.Insert a name");
        printf("\n2.Delete a name");
        printf("\n3.Display the list");
        printf("\n4.Exit");
        printf("\nEnter your choice :");
        scanf("%d", &choice);
        switch (choice)
        {
```

```
        case 1:
          printf("Enter the name to be inserted : ");
          scanf("%s", name);
          insert(name);
          break;
        case 2:
          printf("Enter the name to be deleted : ");
          scanf("%s", name);
          delete (name);
          break;
        case 3:
          display();
          break;
        case 4:
          exit(1);
        default:
          printf("Wrong choice ");
        }
    }
}
```

OUTPUT:

```
1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :1
Enter the name to be inserted : Amar

1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :1
Enter the name to be inserted : Akbar

1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :Antony
Enter the name to be inserted :
1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :3
```

```
Names in the List: Akbar Amar Antony
1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :2
Enter the name to be deleted : amar
Given name not found!
1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :3

Names in the List: Akbar Amar Antony
1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :2
Enter the name to be deleted : Akbar

1.Insert a name
2.Delete a name
3.Display the list
4.Exit
Enter your choice :3

Names in the List: Amar Antony
1.Insert a name
2.Delete a name
3.Display the list
4.Exit
```

Program 10 Stack using Linked List

Write a C program to maintain a stack of integers using a linked list implementation method.

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

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

Node *top = NULL;
void push(int item);
int pop();
int peek();
void display();

int main()
{
   int choice, item;
   while(1)
   {
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Peek\n");
      printf("4. Display\n");
      printf("5. Quit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("Enter the item to be pushed: ");
         scanf("%d", &item);
         push(item);
         break;
      case 2:
         item = pop();
         if (item != -1)
         {
            printf("Popped item is: %d\n", item);
         }
         break;
```

```
        case 3:
            item = peek();
            if (item != -1)
            {
                printf("Top item is: %d\n", item);
            }
            break;
        case 4:
            display();
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}

// Function to add an item to stack. It increases top by 1
void push(int item)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = top;
    top = temp;
}

// Function to remove an item from stack. It decreases top by 1
int pop()
{
    if (top == NULL)
    {
        printf("Stack is empty\n");
        return -1;
    }
    Node *temp = top;
    top = top->next;

    int item = temp->data;
    free(temp);
    return item;
}

// Function to return the top from stack without removing it
int peek()
```

```
{
   if (top == NULL)
   {
      printf("Stack is empty\n");
      return -1;
   }
   return top->data;
}

// Function to display the stack
void display()
{
   if (top == NULL)
   {
      printf("Stack is empty\n");
      return;
   }
   Node *temp = top;
   printf("Stack: ");
   while (temp != NULL)
   {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}
```

OUTPUT

```
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 1
Enter the item to be pushed: 1
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 1
Enter the item to be pushed: 2
1. Push
2. Pop
3. Peek
```

```
4. Display
5. Quit
Enter your choice: 1
Enter the item to be pushed: 3
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 4
Stack: 3 2 1
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 2
Popped item is: 3
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 4
Stack: 2 1
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 2
Popped item is: 2
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 4
Stack: 1
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 2
Popped item is: 1
```

```
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 4
Stack is empty
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 1
Enter the item to be pushed: 5
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 4
Stack: 5
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 1
Enter the item to be pushed: 5
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 4
Stack: 5 5
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 1
Enter the item to be pushed: 6
1. Push
2. Pop
3. Peek
4. Display
```

```
5. Quit
Enter your choice: 4
Stack: 6 5 5
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 2
Popped item is: 6
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your choice: 4
Stack: 5 5
1. Push
2. Pop
3. Peek
4. Display
5. Quit
```

Program 11 Doubly Linked List implementation

Write a C program to support the following operations on a doubly linked list.

a) Insert a new node to the left of the node whose key value is read as an input.

b) Delete a node with given data, if it is found otherwise display appropriate error message.

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

typedef struct node
{
   int data;
   struct node *next;
   struct node *prev;
} Node;

Node *head = NULL;

void insert(int data, int key);
void delete(int data);
void display();

int main(){
   int choice, value, key = -1;
   while (1)
   {
      printf("\n***** MENU *****\n");
      printf("\n1. Insert");
      printf("\n2. Delete");
      printf("\n3. Display");
      printf("\n4. Exit");
      printf("\nEnter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("Enter the value to be inserted: ");
         scanf("%d", &value);
         if(head !=NULL)  // For 1st insertion key not required
         {
            printf("Enter the key: ");
            scanf("%d", &key);
         }
         insert(value, key);
         break;
      case 2:
         printf("Enter the value to be deleted: ");
```

```
        scanf("%d", &value);
        delete(value);
        break;
      case 3:
        display();
        break;
      case 4:
        exit(0);
      default:
        printf("Invalid choice\n");
      }
  }
}


void insert(int data, int key)
{
  Node *newNode = (Node *)malloc(sizeof(Node));
  newNode->data = data;
  newNode->next = NULL;
  newNode->prev = NULL;
  if (head == NULL)
  {
    head = newNode;
    return;
  }

  Node *current = head;

  while (current != NULL && current->data != key)
    current = current->next;

  if (current == NULL)
  {
    printf("Key not found\n");
    return;
  }

  newNode->prev = current->prev;
  current->prev = newNode;

  newNode->next = current;

  if (newNode->prev != NULL)
    newNode->prev->next = newNode;
  else
    head = newNode;
```

```
}

void delete(int data)
{
   Node *temp = head, *prev = NULL, *next = NULL;

   while (temp != NULL && temp->data != data)
   {
      prev = temp;
      temp = temp->next;
   }

   if (temp == NULL)
   {
      printf("Data not found.\n");
      return;
   }
   next = temp->next;

   if (prev == NULL)
      head = next;
   else
      prev->next = next;

   if (next != NULL)
      next->prev = prev;

   free(temp);
}

void display()
{
   Node *ptr = head;
   if(head == NULL)
   {
         printf("List is empty\n");
   }

   printf("\nList: ");
   while (ptr != NULL)
   {
      printf("%d ", ptr->data);
      ptr = ptr->next;
   }
}
```

OUTPUT

```
***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 10
Enter the key: 10


***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 20
Enter the key: 30
Key not found


***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

List: 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 20
Enter the key: 10


***** MENU *****
1. Insert
2. Delete
3. Display
```

```
4. Exit
Enter your choice: 3

List: 20 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 15
Enter the key: 10


***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

List: 20 15 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 18
Enter the key: 3
Key not found


***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

List: 20 15 10

***** MENU *****
1. Insert
```

```
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 18
Enter the key: 15


***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

List: 20 18 15 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Enter the value to be deleted: 18


***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

List: 20 15 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Enter the value to be deleted: 100
Data not found


***** MENU *****
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

List: 20 15 10
```

Program 12 Binary Search Tree implementation

Write a C program

a) To construct a binary search tree of integers.

b) To traverse the tree using inorder, preorder and postorder traversal methods

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

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

// Pointer to the root node
Node *root = NULL;

Node *newNode(int data)
{
   Node *temp = (Node *)malloc(sizeof(Node));
   temp->data = data;
   temp->left = temp->right = NULL;

   return temp;
}

Node *insert(Node *node, int data)
{
   if (node == NULL)
      return newNode(data);

   if (data < node->data)
      node->left = insert(node->left, data);
   else  // Allows duplication of data
      node->right = insert(node->right, data);

   return node;
}

void inorder(Node *root)
{
   if (root != NULL)
```

```
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(Node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(Node *root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

int main()
{
    int choice, value;
    while (1)
    {
        printf("\n1.Insert an element ");
        printf("\n2.Inorder traversal ");
        printf("\n3.Preorder traversal ");
        printf("\n4.Postorder traversal ");
        printf("\n5.Exit ");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the data : ");
            scanf("%d", &value);
            root = insert(root, value);
            break;
        case 2:
            printf("\nInorder traversal:");
            inorder(root);
            break;
        case 3:
             printf("\nPreorder traversal:");
            preorder(root);
            break;
        case 4:
             printf("\nPostorder traversal:");
            postorder(root);
            break;
        case 5:
            exit(1);
        default:
            printf("Wrong choice ");
        }
    }
}
```

OUTPUT

```
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 1
Enter the data : 20

1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 1
Enter the data : 10

1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 1
Enter the data : 30
```

```
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 2
Inorder traversal:10 20 30
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 3
Preorder traversal:20 10 30
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 4
Postorder traversal:10 30 20
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 1
Enter the data : 5

1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 1
Enter the data : 15

1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
```

```
Enter your choice : 2
Inorder traversal:5 10 15 20 30
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 3
Preorder traversal:20 10 5 15 30
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice : 4
Postorder traversal:5 15 10 30 20
1.Insert an element
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
```