DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DATA STRUCTURES AND ALGORITHM

CS 104

LAB FILE

SUBMITTED TO

MR. JATIN SHARMA

SUBMITTED BY:

SAMIR GUPTA
23/CS/498

INDEX

SNO.	OBJECTIVE	DATE	SIGN
1.	Write a program to Implement Linear Search in the C/C++ programming language.	23.02.24	
2.	Write a program to Implement Binary Search in the C/C++ programming language. Assume the list is already sorted.	23.02.24	
3.	Write a program to insert an element at the mid-position in the One-dimensional array.	23.02.24	
4.	Write a program to delete a given row in the two-dimensional array.	23.02.24	
5.	Write a program to implement a stack data structure and perform its operations.	23.02.24	
6.	Write a program to implement two stacks using a single array.	23.02.24	
7.	Write a program to reverse a 5-digit number	01.03.24	
8.	Write a program to convert decimal to binary and vice versa.	01.03.24	

EXPERIMENT-1

AIM - Write a program to Implement Linear Search in the C/C++ programming language.

ALGORITHM -

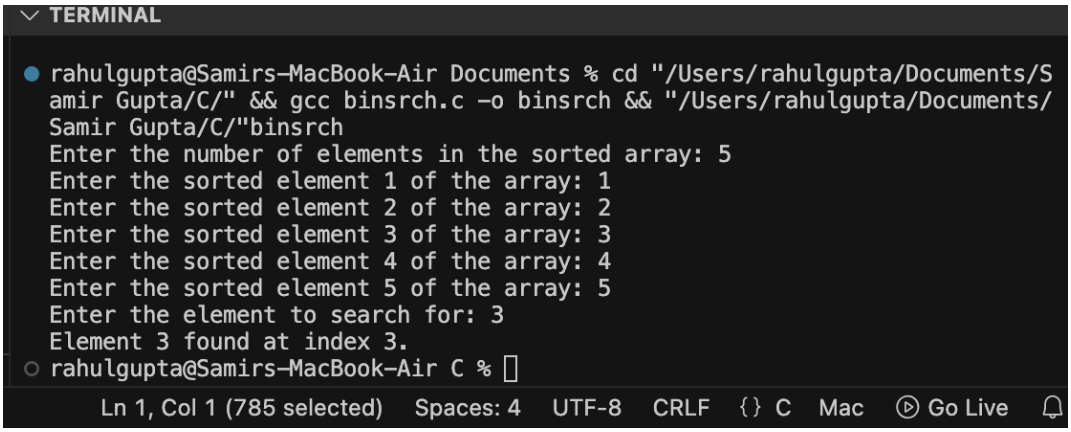
1. Select the first element of the array.
2. Compare the target element with the selected element.
3. If the target element is found, return the index.
4. If the target element is not found after iterating through the entire array, return -1.

CODE -

```
#include <stdio.h>
int linearSearch(int arr[], int n, int target) {
   for (int i = 0; i < n; i++) {
      if (arr[i] == target) {
         return i;
      }
   }
   return -1;
}
int main() {
   int n, target;
   printf("Enter the number of elements in the array: ");
   scanf("%d", &n);
   int arr[n];
   for (int i = 0; i < n; i++) {
      printf("Enter the element %d of the array: ", i+1);
      scanf("%d", &arr[i]);
   }
   printf("Enter the element to search for: ");
   scanf("%d", &target);
   int result = linearSearch(arr, n, target);
   if (result != -1) {
      printf("Element %d found at index %d.\n", target, result);
   } else {
```

```
        printf("Element %d not found in the array.\n", target);
    }
    return 0;
}
```

Output:



EXPERIMENT-2

AIM - Write a program to Implement Binary Search in the C/C++ programming language. Assume the list is already sorted.

ALGORITHM -

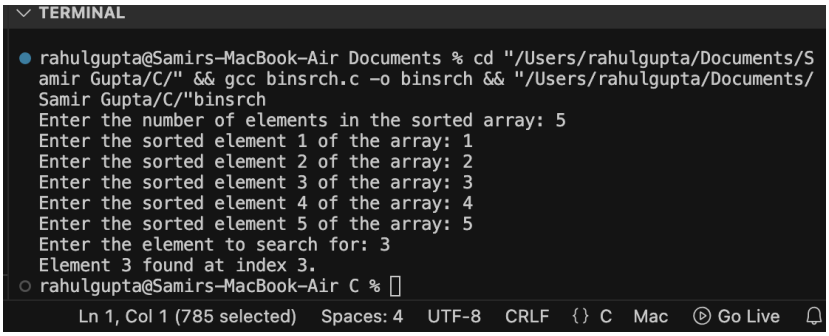
1. Find the middle element of the sorted array.
2. If the middle element is the required element, return its index.
3. If the target is less than the middle element, repeat the search on the left half of the array.
4. If the target is greater than the middle element, repeat the search on the right half of the array.
5. Continue this process until the target is found.

CODE -

```
#include <stdio.h>
int binarySearch(int arr[], int left, int right, int target) {
  if (left <= right) {
      int mid = left + (right - left) / 2;
      if (arr[mid] == target) {
          return mid;
      }
      if (arr[mid] > target) {
          return binarySearch(arr, left, mid - 1, target);
      }
      return binarySearch(arr, mid + 1, right, target);
  }
  return -1;
}
int main() {
  int n, target;
  printf("Enter the number of elements in the sorted array: ");
  scanf("%d", &n);
  int arr[n];
```

```
    for (int i = 0; i < n; i++) {
        printf("Enter the sorted element %d of the array: ", i+1);
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search for: ");
    scanf("%d", &target);
    int result = binarySearch(arr, 0, n - 1, target);
    if (result != -1) {
        printf("Element %d found at index %d.\n", target, result+1);
    } else {
        printf("Element %d not found in the array.\n", target);
    }
    return 0;
}
```

Output:



EXPERIMENT-3

AIM - Write a program to insert an element at the mid-position in the One-dimensional array.

ALGORITHM -

1. Calculate the mid-position of the array.
2. Shift elements from the mid-position to the end of the array one position to the right.
3. Insert the new element at the mid-position.

CODE -

```
#include<stdio.h>
void display(int arr[],int n){
   for(int i = 0; i < n;i++){
      printf("%d\n",arr[i]);
   }
   printf("\n");
}

int indInsertion(int arr[],int size, int element, int capacity, int index){
   if(size>=capacity){
      return -1;
   }
   for(int i = size-1;i>=index;i--){
      arr[i+1]=arr[i];
   }
   arr[index]=element;
   return 1;
}
int main(){
   int arr[100]={7, 8, 12, 17, 88};
   int size = 5, element = 45, index=4;
   display(arr , size);
   indInsertion(arr , size, element, 100, index);
   size+=1;
   display(arr, size);

   return 0;
```

}

OUTPUT -


pta/Documents/Samir Gupta/DSA(2nd sem)/programs
rayInsertoin.c -o ArrayInsertoin && "/Users/ra
ments/Samir Gupta/DSA(2nd sem)/programs/"Array
7
8
12
17
88
7
8
12
17
45
88

EXPERIMENT-4

AIM - Write a program to delete a given row in the two-dimensional array.

ALGORITHM -

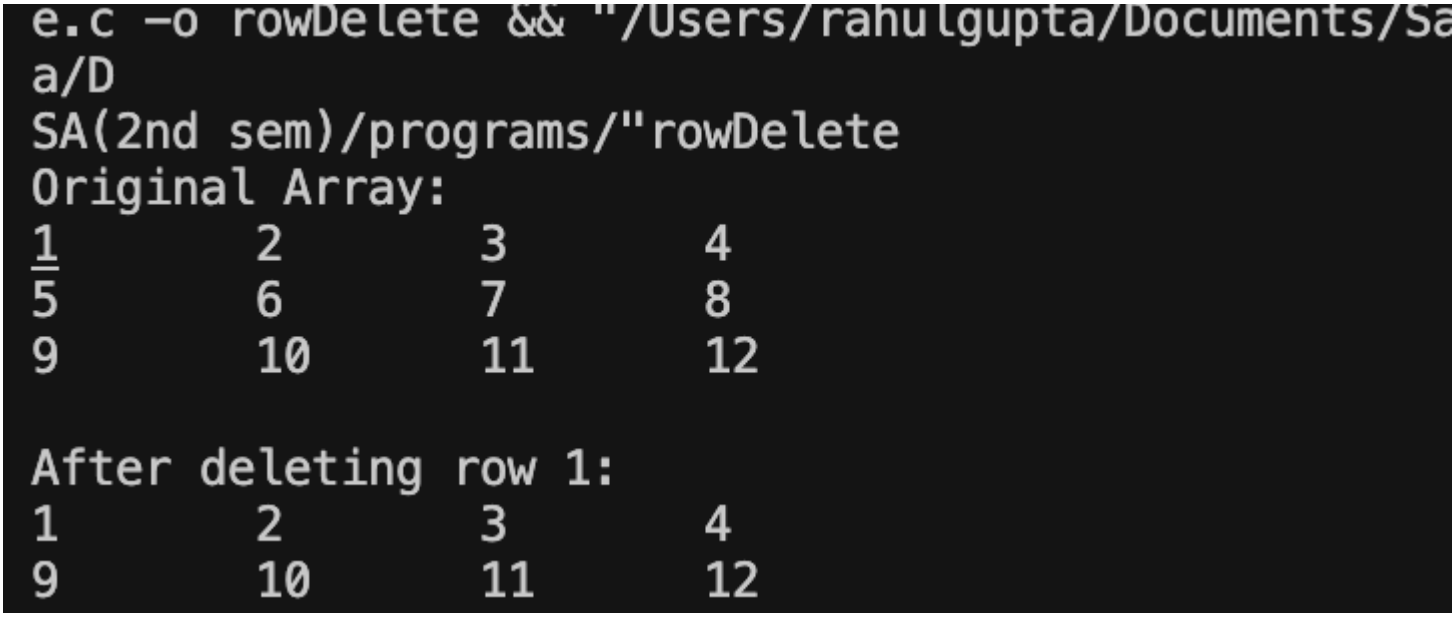
1. Move all rows below the deleted row one position up.
2. Decrement the total number of rows.

CODE -

```
#include<stdio.h>
#define ROW 3
#define COL 4
void display(int arr[][COL],int rows){
   for(int i = 0; i < rows; i++){
      for(int j=0; j < COL; j++){
         printf("%d\t",arr[i][j]);
      }
      printf("\n");
   }
}
void deleteRow(int arr[][COL],int *rows, int index){
   if(index<0 || index >= *rows){
      printf("Invalid row index\n");
      return;
   }
   for(int i = index; i<*rows-1; i++){
      for(int j=0; j<COL; j++){
         arr[i][j] = arr[i+1][j];
      }
   }
   (*rows)--;
}
int main(){
   int rows = ROW;
   int arr[ROW][COL]= {{1, 2, 3, 4},
                       {5, 6, 7, 8},
                       {9, 10, 11, 12}};
   printf("Original Array:\n");
   display(arr, rows);
```

```
    deleteRow(arr, &rows, 1);
    printf("\nAfter deleting row 1:\n");
    display(arr, rows);
    return 0;
}
```

OUTPUT -



EXPERIMENT-5

AIM - Write a program to implement a stack data structure and perform its operations.

ALGORITHM -

1. Initialize a stack data structure.
2. Implement push operation to insert elements onto the stack.
3. Implement pop operation to remove elements from the stack.
4. Implement peek operation to view the top element of the stack.
5. Implement isEmpty operation to check if the stack is empty.
6. Implement isFull operation to check if the stack is full.

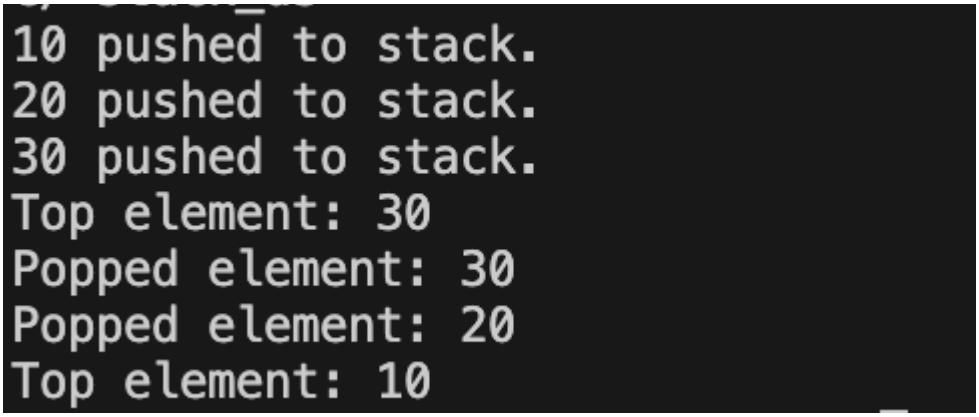
CODE -

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_SIZE 100
typedef struct {
    int data[MAX_SIZE];
    int top;
} Stack;
void initStack(Stack *s) {
    s->top = -1;}
bool isEmpty(Stack *s) {
    return s->top == -1;}
bool isFull(Stack *s) {
    return s->top == MAX_SIZE - 1;}
void push(Stack *s, int element) {
    if (!isFull(s)) {
```

```
        s->data[++s->top] = element;
        printf("%d pushed to stack.\n", element);
    } else {
        printf("Stack overflow! Unable to push %d\n", element);
    }
}
int pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->data[s->top--];
    } else {
        printf("Stack underflow! Unable to pop.\n");
        return -1;
    }
}
int peek(Stack *s) {
    if (!isEmpty(s)) {
        return s->data[s->top];
    } else {
        printf("Stack is empty!\n");
        return -1;
    }
}
int main() {
    Stack s;
    initStack(&s);
    push(&s, 10);
    push(&s, 20);
    push(&s, 30);
    printf("Top element: %d\n", peek(&s));
    printf("Popped element: %d\n", pop(&s));
    printf("Popped element: %d\n", pop(&s));
    printf("Top element: %d\n", peek(&s));
    return 0;
```

}

OUTPUT -


10 pushed to stack.
20 pushed to stack.
30 pushed to stack.
Top element: 30
Popped element: 30
Popped element: 20
Top element: 10

EXPERIMENT-6

AIM - Write a program to implement two stacks using a single array.

ALGORITHM -

1. Divide the array into two halves to represent two stacks.
2. Implement push and pop operations for each stack separately.
3. Keep track of the top index of each stack

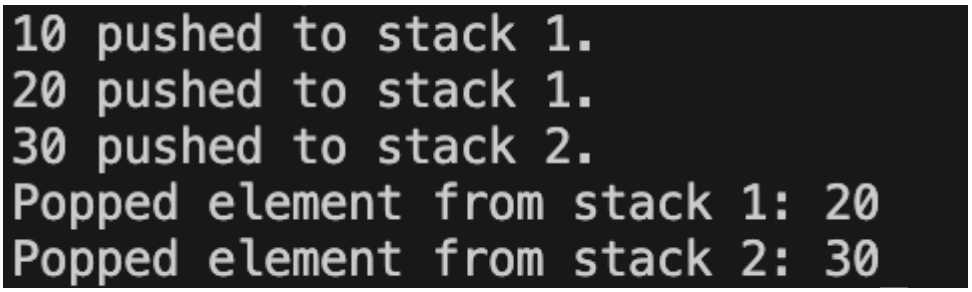
CODE -

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_SIZE 100
typedef struct {
    int data[MAX_SIZE];
    int top1;
    int top2;
} TwoStacks;
void initTwoStacks(TwoStacks *ts) {
    ts->top1 = -1; // Top of stack 1
    ts->top2 = MAX_SIZE; // Top of stack 2
}
bool isFull(TwoStacks *ts) {
    return ts->top1 == ts->top2 - 1;
}
bool isEmpty1(TwoStacks *ts) {
    return ts->top1 == -1;
}
bool isEmpty2(TwoStacks *ts) {
    return ts->top2 == MAX_SIZE;
}
```

```
void push1(TwoStacks *ts, int element) {
    if (!isFull(ts)) {
        ts->data[++ts->top1] = element;
        printf("%d pushed to stack 1.\n", element);
    } else {
        printf("Stack 1 overflow! Unable to push %d\n", element);
    }
}
void push2(TwoStacks *ts, int element) {
    if (!isFull(ts)) {
        ts->data[--ts->top2] = element;
        printf("%d pushed to stack 2.\n", element);
    } else {
        printf("Stack 2 overflow! Unable to push %d\n", element);
    }
}\
int pop1(TwoStacks *ts) {
    if (!isEmpty1(ts)) {
        return ts->data[ts->top1--];
    } else {
        printf("Stack 1 underflow! Unable to pop.\n");
        return -1;
    }
}
int pop2(TwoStacks *ts) {
    if (!isEmpty2(ts)) {
        return ts->data[ts->top2++];
    } else {
        printf("Stack 2 underflow! Unable to pop.\n");
        return -1;
    }
}
int main() {
```

```
    TwoStacks ts;
    initTwoStacks(&ts);
    push1(&ts, 10);
    push1(&ts, 20);
    push2(&ts, 30);
    printf("Popped element from stack 1: %d\n", pop1(&ts));
    printf("Popped element from stack 2: %d\n", pop2(&ts));
    return 0;
}
```

OUTPUT -



EXPERIMENT-7

AIM - Write a program to reverse a 5-digit number

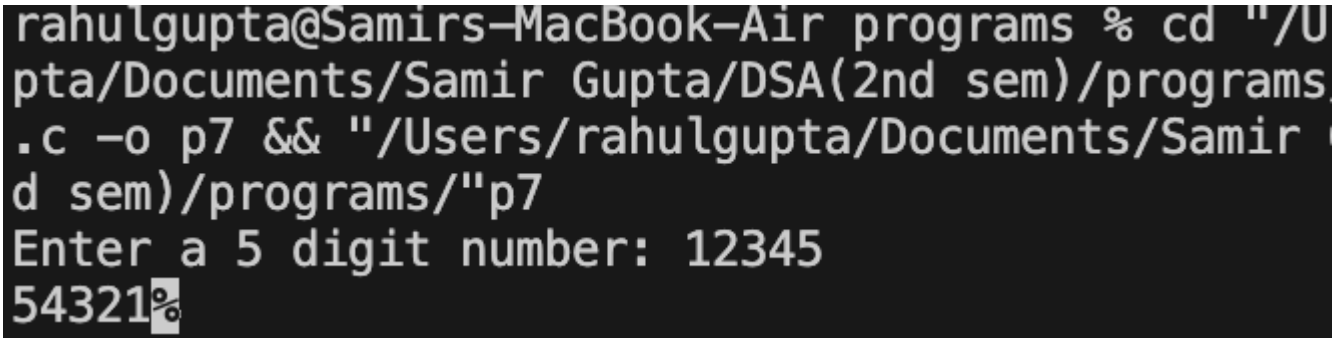
ALGORITHM -

1. Extract each digit of the 5-digit number iteratively.
2. Reconstruct the reversed number by appending the digits in reverse order.
3. Display the reversed number.

CODE -

```
#include<stdio.h>
int reverse(int num){
   int rev=0;
   while(num!=0){
      rev=rev*10+num%10;
      num=num/10;
   }
   return rev;
}
int main(){
   int num,rev;
   printf("Enter a 5 digit number: ");
   scanf("%d",&num);
   rev=reverse(num);
   printf("%d",rev);
   return 0;
}
```

OUTPUT -


rahulgupta@Samirs-MacBook-Air programs % cd "/U
pta/Documents/Samir Gupta/DSA(2nd sem)/programs
.c -o p7 && "/Users/rahulgupta/Documents/Samir
d sem)/programs/"p7
Enter a 5 digit number: 12345
54321%

EXPERIMENT-8

AIM - Write a program to convert decimal to binary and vice versa.

ALGORITHM -

- Algorithm (Decimal to Binary):

1. Initialize variables to hold binary number and remainder.
2. Perform repeated division of the decimal number by 2.
3. Record the remainders to obtain the binary equivalent.
4. Reverse the binary equivalent to get the final binary number.
5. Display the binary number.

- Algorithm (Binary to Decimal):

1. Initialize variables to hold decimal number, base, and remainder.
2. Perform the conversion by multiplying each binary digit by powers of 2.
3. Sum the results to obtain the decimal equivalent.
4. Display the decimal number.

CODE -

- **Decimal to Binary**

```
#include <stdio.h>

long decimalToBinary(int decimal) {
    long binary = 0;
    int remainder, base = 1;
```

```
    while (decimal > 0) {
        remainder = decimal % 2;
        binary += remainder * base;
        decimal /= 2;
        base *= 10;
    }

    return binary;
}

int main() {
    int decimalNumber;
    printf("Enter a decimal number: ");
    scanf("%d", &decimalNumber);

    long binaryNumber = decimalToBinary(decimalNumber);
    printf("Binary equivalent: %ld\n", binaryNumber);

    return 0;
}
```

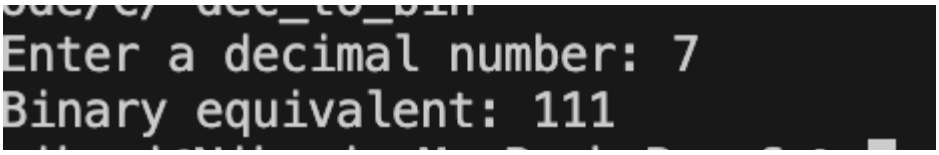
- **Binary to Decimal**

```
#include <stdio.h>
int binaryToDecimal(long binary) {
    int decimal = 0, base = 1, remainder;
    while (binary > 0) {
        remainder = binary % 10;
        decimal += remainder * base;
        binary /= 10;
        base *= 2;
    }
    return decimal;
```

```
}
int main() {
    long binaryInput;
    printf("Enter a binary number: ");
    scanf("%ld", &binaryInput);
    int decimalResult = binaryToDecimal(binaryInput);
    printf("Decimal equivalent: %d\n", decimalResult);
    return 0;
}
```

OUTPUT -

- **Decimal to Binary**



- **Binary to Decimal**