DSA PRACTICALS

WEEK - 1

PRACTICAL - 1 : WAP to demonstrate the use of dynamic memory allocation. DMA functions malloc(),calloc(), Realloc, free().

ALGORITHM :

1) Include the necessary header file for memory allocation: #include <stdlib.h>
2) Declare variables or data structures to hold the allocated memory.
3) Allocate memory using malloc().
4) Allocate memory using calloc().
5) Use the allocated memory as needed.
6) Reallocate memory using realloc().
7) Use the reallocated memory.
8) Free the allocated memory using free().
9) After freeing the memory, ensure that the pointers are set to NULL to avoid using invalid memory references.

PROGRAM :

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

int main()
{
   int n, i;
   int *arr1, *arr2;
   printf("Enter the size of the array: ");
   scanf("%d", &n);
   arr1 = (int *)malloc(n * sizeof(int));
   printf("Enter %d integers:\n", n);
   for (i = 0; i < n; i++)
   {
       scanf("%d", &arr1[i]);
   }
   printf("The array you entered is:\n");
   for (i = 0; i < n; i++)
   {
       printf("%d ", arr1[i]);
   }
```

```
    printf("\n");
    free(arr1);
    printf("\nEnter the size of the array: ");
    scanf("%d", &n);
    arr2 = (int *)calloc(n, sizeof(int));
    printf("Enter %d integers:\n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &arr2[i]);
    }
    printf("The array you entered is:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d ", arr2[i]);
    }
    printf("\n");
    free(arr2);
    int *arr3 = (int *)malloc(5 * sizeof(int));
    printf("\nInitial array of size 5:\n");
    for (i = 0; i < 5; i++)
    {
        arr3[i] = i + 1;
        printf("%d ", arr3[i]);
    }
    arr3 = (int *)realloc(arr3, 8 * sizeof(int));
    printf("\nArray after reallocating to size 8:\n");
    for (i = 0; i < 8; i++)
    {
        printf("%d ", arr3[i]);
    }
    printf("\n");
    free(arr3);
    return 0;
}
```

OUTPUT :

```
Enter the size of the array: 4
Enter 4 integers:
1 2 3 4
The array you entered is:
1 2 3 4
```

```
Enter the size of the array: 3
Enter 3 integers:
5 6 7
The array you entered is:
5 6 7

Initial array of size 5:
1 2 3 4 5
Array after reallocating to size 8:
1 2 3 4 5 0 0 0
```

WEEK - 2

PRACTICAL - 1 : WAP to insert the group of numbers at the given position in an array.

ALGORITHM :

```
1) Start
2) Read the size of the array (size)
3) Read the position at which the group should be inserted (pos)
4) Read the size of the group of numbers (groupSize)
5) Create an array of size (size + groupSize) to accommodate the new group of numbers
6) Read the elements of the original array of size (size)
7) Read the elements of the group to be inserted (group)
8) Shift the elements in the original array to the right from position (pos) to (size - 1)
by (groupSize) positions
        a. Start from the last element of the original array and move towards the position (pos)
        b. Move each element to the right by (groupSize) positions
9) Insert the group of numbers into the original array at position (pos)

    ● Start from position (pos) and move towards position (pos + groupSize - 1)
    ● Replace each element with the corresponding element from the group of
      numbers

10) Print the modified array
11) Stop
```

PROGRAM :

```
#include<stdio.h>
int main()
{
   int arr[10],i,num,pos;
```

```
    printf("Enter the array elements.");
    for(i=0;i<10;i++)
    {
        scanf("%d",&arr[i]);
    }
    printf("The array elements are:");

    for(i=0;i<10;i++)
    {
        printf("%d",arr[i]);
        printf("\n");
    }

    printf("Enter the value to insert.");
    scanf("%d",&num);

    printf("Enter the Position where to insert.");
    scanf("%d",&pos);

    for(i=10;i>=pos;i--)
    {
        arr[i]=arr[i-1];
    }
    arr[i]=num;

    printf("The array Elements after inserting.\n");
    for(i=0;i<10;i++)
    {
        printf("%d\n ",arr[i]);
    }
}
```

OUTPUT :

```
Array before insertion: 1 2 3 4 5
Array after insertion: 1 2 6 7 8 3 4 5
```

PRACTICAL - 2 : WAP to replace a substring with another substring in the given string.

ALGORITHM :

1) Start with the input string, the substring to be replaced, and the new substring.
2) Find the position/index of the first occurrence of the substring in the input string. If the substring is not found, exit or handled accordingly.
3) Create a new string or character array to hold the modified string.
4) Copy the characters from the beginning of the input string up to the position of the substring into the new string.
5) Append the new substring to the new string.
6) Skip the characters in the input string corresponding to the length of the substring that needs to be replaced.
7) Copy the remaining characters from the input string after the position of the substring into the new string.
8) The new string now contains the modified string with the replaced substring.

PROGRAM :

```c
#include <stdio.h>
int main()
{
   char str[200], pat[20], new_str[200], rep_pat[100];
   int i=0, j=0, k, n=0, copy_loop=0, rep_index=0;
   printf("\n Enter the string : ");
   scanf("%s",str);
   printf("\n Enter the pattern to be replaced: ");
   scanf("%s",pat);
   printf("\n Enter the replacing pattern: ");
   scanf("%s",rep_pat);
   while(str[i]!='\0')
   {
       j=0,k=i;
       while(str[k]==pat[j] && pat[j]!='\0')
       { k++; j++; }
       if(pat[j]=='\0')
       {
           copy_loop=k;
           {
               new_str[n] = rep_pat[rep_index];
               rep_index++;
               n++;
```

```
            }
        }
        new_str[n] = str[copy_loop];
        i++;
        copy_loop++;
        n++;
    }
    new_str[n]='\0';
    printf("\n The new string is : %s",new_str);

}
```

OUTPUT :

```
Enter a string: hello world
Enter a substring to replace: world
Enter the new substring: there
Resulting string: hello there
```

PRACTICAL - 3 : WAP to sort the names in ascending order.

ALGORITHM :

1) Create an array to hold the names.
2) Read the names into the array.
3) Use a sorting algorithm like bubble sort, selection sort, or insertion sort to sort the names in ascending order.
4) Compare each pair of adjacent names and swap them if they are out of order.
5) Repeat the comparison and swapping process until the entire array is sorted. 6) After the sorting is complete, the names will be in ascending order.

PROGRAM :

```
#include <stdio.h>
#include <string.h>
#define MAX_NAMES 10

int main()
{
    char names[MAX_NAMES][100], temp[100];
```

```c
    int i, j, n;
    printf("Enter the number of names (up to %d): ", MAX_NAMES); scanf("%d", &n);
    printf("Enter %d names:\n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%s", names[i]);
    }
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (strcmp(names[j], names[j+1]) > 0)
            {
                strcpy(temp, names[j]); strcpy(names[j], names[j+1]);
strcpy(names[j+1], temp);
            }
        }
    }
    printf("Sorted names in ascending order:\n");
    for (i = 0; i < n; i++)
    {
        printf("%s\n", names[i]);
    }
    return 0;
}
```

OUTPUT :

```
Enter the number of names (up to 10): 5
Enter 5 names:
Alice
Charlie
Eve
Bob
David

Sorted names in ascending order:
Alice
Bob
Charlie
David
Eve
```

PRACTICAL - 4 : WAP to create a database of 5 students for 3 subjects using arrays.

ALGORITHM :

1) **Create arrays to store the student data. You will need an array for each subject and for each student's name.**
2) **Read the data for each student and each subject.**
3) **Store the data in the respective arrays.**
4) **Repeat steps 2 and 3 for all the students and subjects.**
5) **Access the stored data as needed for further processing or display.**

PROGRAM :

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

int main()
{
   int marks[5][3],high,sum,i=0,j=0;
   string nam[5];

   cout<<"Enter the data required : "<<endl;

   for(i=0;i<5;i++)
   {
       cout<<"Enter the name of the student :";
       cin>>nam[i];

       cout<<"Enter the marks of C++,DSA and DE :";

       for(j=0;j<3;j++)
       {
           cin>>marks[i][j];
       }
   }

   cout<<"Here are the results:";

   for(i=0;i<5;i++)
   {
       j=0;
       high = marks[i][j];
       for(j=0;j<3;j++)
       {
           sum = sum + marks[i][j];
           for(int k=0;k<2;k++)
           if(high<marks[i][k+1])
           {
               high = marks[i][k+1];
           }
       }
       float total = sum/3;
       cout<<endl<<nam[i] <<" gets : "<<total<<"%"<<endl;
       sum=0;
       cout<<"Highest marks : "<<high<<endl;
   }
}
```

OUTPUT :

```
Enter the marks for student 1:
Subject 1: 70
Subject 2: 80
Subject 3: 90
Enter the marks for student 2:
Subject 1: 80
Subject 2: 90
Subject 3: 100
Enter the marks for student 3:
Subject 1: 60
Subject 2: 70
Subject 3: 80
Enter the marks for student 4:
Subject 1: 50
Subject 2: 60
Subject 3: 70
Enter the marks for student 5:
Subject 1: 90
Subject 2: 80
Subject 3: 70

Results:
Student     Percent      Max Marks (Subject 1, Subject 2, Subject 3)
1             80.00          90, 90, 100
2             90.00          90, 90, 100
3             70.00          90, 80, 90
4             60.00          90, 80, 70
```

WEEK - 3

<u>PRACTICAL - 1</u> : WAP to perform Push, Pop, and Peek operations on a stack.

ALGORITHM :

1) Create a stack data structure with an underlying array and a variable to keep track of the top of the stack.
2) Initialise the top of the stack to -1, indicating an empty stack.
3) Implement a push operation:
  a. Check if the stack is full.
  b. If not full, increment the top of the stack by 1. c. Insert the new element at the top of the stack.
4) Implement a pop operation:
  a. Check if the stack is empty.
  b. If not empty, retrieve the element at the top of the stack. c. Decrement the top of the stack by 1.
5) Implement a peek operation:
  a. Check if the stack is empty.
  b. If not empty, retrieve and return the element at the top of the stack.

PROGRAM :

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

#define max 10
int top = -1;

class stack
{
public:
   int st[max];

public:
   int isEmpty()
   {
       if (top == -1)
       {
           return 1;
       }
       else
       {
           return 0;
       }
   }
   int isFull()
```

```cpp
    {
        if (top == max-1)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }

    void push(int a)
    {
        if(isFull())
        {
            cout << "The stack is full.";
        }
        else
        {
            top++;
            st[top] = a;
        }
    }

    void pop()
    {
        if (isEmpty())
        {
            cout << "The stack is Empty.";
        }
        else
        {
            st[top] = 0;
            top--;
        }
    }

    void print()
    {
        for (int i = 0; i <= top; i++)
        {
            cout << st[i] << endl;
        }
};
```

```
int main()
{
   stack s;

   s.push(1);
   s.push(2);
   s.push(3);
   s.push(4);
   s.push(5);
   s.push(6);
   s.push(7);
   s.push(8);
   s.push(9);

   s.print();
   cout<<endl;

   s.pop();
   s.pop();
   s.pop();
   s.pop();
   s.pop();

   s.print();
}
```

OUTPUT :

```
1
2
3
4
5
6
7
8
9

1
2
3
4
```

WEEK - 4

PRACTICAL - 1 : Implement a program to convert infix notation to postfix notation using stack.

ALGORITHM :

1) Create an empty stack to hold operators.
2) Create an empty list to store the output in postfix notation.
3) Scan the infix expression from left to right.
4) If the current token is an operand (number or variable), append it to the output list. 5) If the current token is a left parenthesis '(', push it onto the stack.
6) If the current token is an operator (+, -, *, /, etc.), do the following: a. While there are operators at the top of the stack with greater precedence or equal precedence and left associativity, pop them from the stack and append them to the output list. b. Push the current operator onto the stack.
7) If the current token is a right parenthesis ')', do the following: a. While the top of the stack is not a left parenthesis, pop operators from the stack and append them to the output list. b. Pop the left parenthesis from the stack and discard it.
8) After scanning the entire infix expression, pop any remaining operators from the stack and append them to the output list.
9) The output list now contains the infix expression converted to postfix notation.

PROGRAM :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_EXPRESSION_LENGTH 100

struct stack
{
   char data[MAX_EXPRESSION_LENGTH];
   int top;
};
void push(struct stack *s, char c)
{
   if (s->top == MAX_EXPRESSION_LENGTH - 1)
   {
       printf("Error: Stack overflow\n");
       exit(EXIT_FAILURE);
   }
   s->top++;
   s->data[s->top] = c;
```

```
}
char pop(struct stack *s)
{
   if (s->top == -1)
   {
       printf("Error: Stack underflow\n");
       exit(EXIT_FAILURE);
   }
   char c = s->data[s->top];
   s->top--;
   return c;
}
int precedence(char op)
{
   if (op == '+' || op == '-')
   {
       return 1;
   }
   else if (op == '*' || op == '/')
   {
       return 2;
   }
   else if (op == '^')
   {
       return 3;
   }
   else
   {
       return -1;
   }
}
int is_operator(char c)
{
   return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}
void infix_to_postfix(char *infix, char *postfix)
{
   struct stack s;
   s.top = -1;
   int i, j;
   for (i = 0, j = 0; infix[i] != '\0'; i++)
   {
       if (infix[i] == '(')
       {
           push(&s, infix[i]);
```

```
        }
        else if (infix[i] == ')')
        {
            while (s.top != -1 && s.data[s.top] != '(')
            {
                postfix[j++] = pop(&s);
            }
            if (s.top == -1)
            {
                printf("Error: Mismatched parentheses\n");
                exit(EXIT_FAILURE);
            }
            pop(&s);
        }
        else if (is_operator(infix[i]))
        {
            while (s.top != -1 && precedence(infix[i]) <= precedence(s.data[s.top]))
            {
                postfix[j++] = pop(&s);
            }
            push(&s, infix[i]);
        }
        else
        {
            postfix[j++] = infix[i];
        }
    }
    while (s.top != -1)
    {
        if (s.data[s.top] == '(')
        {
            printf("Error: Mismatched parentheses\n");
            exit(EXIT_FAILURE);
        }
        postfix[j++] = pop(&s);
    }
    postfix[j] = '\0';
}
int main()
{
    char infix[MAX_EXPRESSION_LENGTH];
    char postfix[MAX_EXPRESSION_LENGTH];
    printf("Enter infix expression: ");
    fgets(infix, MAX_EXPRESSION_LENGTH, stdin);
    infix[strlen(infix) - 1] = '\0'; // remove newline character
    infix_to_postfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

OUTPUT :

```
Enter infix expression: 3+4*5/(6-2)
Postfix expression: 345*62-/+
```

PRACTICAL - 2 : Write a program to implement QUEUE using arrays that performs following operations:

a) INSERT
b) DELETE
c) DISPLAY

ALGORITHM :

```
1) Initialise an empty array with a fixed size and set front and rear pointers to -1.
2) INSERT operation:
        a. Check if the queue is full by comparing rear with the array size.
        b. If the queue is full, display an overflow message.
        c. If the queue is not full:
        i. Increment rear by 1.
        ii. Read the element to be inserted.
        iii. Store the element at the rear position in the array.
3) DELETE operation:
        a. Check if the queue is empty by comparing front with rear.
        b. If the queue is empty, display an underflow message.
        c. If the queue is not empty:
        i. Increment front by 1.
        ii. Retrieve and display the element at the front position from the array.
4) DISPLAY operation:
        a. Check if the queue is empty by comparing front with rear.
        b. If the queue is empty, display a message indicating that the queue is empty.
        c. If the queue is not empty:
        i. Start a loop from front + 1 to rear.
        ii. Retrieve and display each element in the array.
5) Repeat steps 2-4 as needed.
```

PROGRAM :

```
#include <stdio.h>
#define MAXSIZE 10

int queue[MAXSIZE];
int front = -1;
int rear = -1;
void insert(int item)
{
   if (rear == MAXSIZE - 1)
   {
       printf("Queue is full\n");
       return;
   }
   else
   {
       if (front == -1)
       {
           front = 0;
       }
       rear++;
       queue[rear] = item;
       printf("Inserted element: %d\n", item);
   }
}
void Delete()
{
   if (front == -1 || front > rear)
   {
       printf("Queue is empty\n");
       return;
   }
   else
   {
       printf("Deleted element: %d\n", queue[front]);
       front++;
       if (front > rear)
       {
           front = rear = -1;
       }
   }
}
void display()
```

```
{
   if (front == -1)
   {
       printf("Queue is empty\n");
   }
   else
   {
       int i;
       printf("Queue elements: ");
       for (i = front; i <= rear; i++)
       {
       OUTPUT:
           printf("%d ", queue[i]);
       }
       printf("\n");
   }
}
int main()
{
   int choice, item;
   while (1)
   {
       printf("1. Insert\n");
       printf("2. Delete\n");
       printf("3. Display\n");
       printf("4. Quit\n");
       printf("Enter your choice: ");
       scanf("%d", &choice);
       switch (choice)
       {
       case 1:
           printf("Enter element to be inserted: ");
           scanf("%d", &item);
           insert(item);
           break;
       case 2:
           Delete ();
           break;
       case 3:
           display();
           break;
       case 4:
           return 0;
       default:
           printf("Invalid choice\n");
```

```
            }
        }
}
```

OUTPUT :

```
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 1
Enter element to be inserted: 10
Inserted element: 10
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 1
Enter element to be inserted: 20
Inserted element: 20
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 1
Enter element to be inserted: 30
Inserted element: 30
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 3
Queue elements: 10 20 30
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 2
Deleted element: 10
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 3
Queue elements: 20 30
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 4
```