DATA STRUCTURE

NAME : M .ASHRITHA

COURSE CODE : CSA0390

DATE : 24/07/24

DAY : 01

1.writing a recursive function to caluculate the factorial of a number .

```
#include <stdio.h>
 int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
int main() {
    int number = 3;
    int result = factorial(number);
    printf("Factorial of %d = %d", number, result);
    return 0;
}
```

<u>output</u> :

```
Factorial of 3 = 6
```

2 . write a c program to find duplicate elements in an array

```
#include <stdio.h>


int main() {
        int arr[] = {1, 2, 3, 4, 2, 7, 8, 8, 3};
        int size = sizeof(arr) / sizeof(arr[0]);
  printf("Duplicate elements in the array are: ");
        for (int i = 0; i < size; i++) {
                for (int j = i + 1; j < size; j++) {
                        if (arr[i] == arr[j]) {
                                printf("%d ", arr[j]);
                                break;
                        }
                }
        }
return 0;
}
```

<u>output</u> :

```
Duplicate elements in the array are : 2 3 4
```

3 . write a c program to find max and min elements from in an array

```
#include <stdio.h>
int main() {
    int arr[] = {10, 5, 8, 20, 15};
    int n = sizeof(arr) / sizeof(arr[0]);
    int max = arr[0];
    int min = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
        if (arr[i] < min) {
            min = arr[i];
        }
    }
    printf("Maximum element in the array: %d\n", max);
    printf("Minimum element in the array: %d\n", min);
return 0;
}
```

output :

```
Maximum element in the array: 20
Minimum element in the array: 5
```

4 . given a number n the task is to print the fibonacci series and the sum of the series using recursion .

```
#include <stdio.h>
int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
    int n, i;
    unsigned long long sum = 0;
    printf("Enter the number of terms: ");
    scanf("%d", &n);
    printf("Fibonacci Series: ");
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
        sum += fibonacci(i);
    }
    printf("\nSum of Fibonacci Series: %llu", sum);
   return 0;
}
```

output :

```
Enter the number of terms: 10
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
Sum of Fibonacci Series: 88
```

5 . you are given an array arr in increasing order. find the elment x from arr using binary.

```
#include <stdio.h>
int binarySearch(int arr[], int left, int right, int x) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] < x)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}
int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14, 16};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n - 1, x);
    if (result == -1)
        printf("Element not found\n");
    else
        printf("Element found at index %d\n", result);
  return 0;
}
```

output :

```
Element found at index : 4
```

6 . write a c program to implement following operations

a)traverse

b)search

c)insert

d)delete

e)update

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

};

void traverse(struct Node* head) {

    struct Node* temp = head;

    while (temp != NULL) {

        printf("%d ", temp->data);

        temp = temp->next;

    }

}

int search(struct Node* head, int key) {

    struct Node* current = head;

    while (current != NULL) {

        if (current->data == key) {

            return 1;

        }

        current = current->next;

    }
    return 0;
}
void insert(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
void delete(struct Node** head_ref, int key) {
    struct Node* temp = *head_ref, *prev;
    if (temp != NULL && temp->data == key) {
        *head_ref = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) return;
    prev->next = temp->next;
    free(temp);
}
void update(struct Node* head, int old_data, int new_data) {
    struct Node* temp = head;
    while (temp != NULL) {
        if (temp->data == old_data) {
            temp->data = new_data;
            return;
        }
        temp = temp->next;
    }
}
int main() {
    struct Node* head = NULL;
    insert(&head, 1);
    insert(&head, 2);
    insert(&head, 3);
    printf("Initial Linked List: ");
    traverse(head);
    printf("\n");
    int key = 2;
    if (search(head, key)) {
        printf("%d found in the Linked List.\n", key);
    } else {
        printf("%d not found in the Linked List.\n", key);
    }
    delete(&head, 2);
    printf("Linked List after deleting 2: ");
    traverse(head);
    printf("\n");
    update(head, 1, 10);
    printf("Linked List after updating 1 to 10: ");
    traverse(head);
    printf("\n");
   return 0;
}
```

output :

```
Initial Linked List: 3 2 1
2 found in the Linked List.
Linked List after deleting 2: 3 1
Linked List after updating 1 to 10: 3 10
```

7. write a c program of linear search.

```
#include <stdio.h>
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}
int main() {
    int arr[] = {2, 4, 6, 8, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 6;

    int result = linearSearch(arr, n, key);

    if (result == -1) {

        printf("Element not found\n");

    } else {

        printf("Element found at index %d\n", result);

    }

    return 0;

}
```

output :

```
Element found at index : 2
```

8 . write a c program of binary search.

```
#include <stdio.h>

int binarySearch(int arr[], int left, int right, int target) {

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {

            return mid;

        }

        if (arr[mid] < target) {

            left = mid + 1;

        } else {

            right = mid - 1;

        }

    }
```

```
        return -1;

}

int main() {

        int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

        int n = sizeof(arr) / sizeof(arr[0]);

        int target = 12;

        int result = binarySearch(arr, 0, n - 1, target);

        if (result == -1) {

                printf("Element not found\n");

        } else {

                printf("Element found at index %d\n", result);

        }

      return 0;

}
```

output :

```
Element found at index :    5
```