```
#include "Day1_Header.h"
#include <stdio.h>
#include <math.h>

int upper_case_to_lower_converter(char letter) { // pper case to lower converter
function defination

    if (letter < 64) // checking if character is in letter or not, Ascii number les
than 64 is of symbols
        return -1;

    else if (letter > 97) // checking if character is in lower case or not, Ascii
number of lower case letter starts from 97, with 97 being 'a'
        return -2;

    else
        return letter + 32;
}

float circle_area_generator(int radius) { // Area of Circle function defination
    float area;
    area = 3.14 * radius * radius;
    return area;
}

float simple_interest_calculator(int princi_amt, int time_period, int rate) { //
Simple interest function defination

    return (princi_amt * time_period * rate) / 100;
}

double compound_interest_calculator(double princi_amt, double time_period, double
rate) { // Compound interest function defination
    double rate_calc, int_calc;

    rate_calc = (1 + rate * 0.01);
    int_calc = PRINCI_AMT * (pow(rate_calc, TIME_PERIOD) - 1);
    return int_calc;
}

int check_even_number(int number) { //Check even number function defination
    if (number % 2 == 0)
        return 1;
    else
        return 0;
}

float temperature_scale_converter(int scale, float temperature) { //Temperature
Scale Converter function defination
    float converted_temp;
```

```
    /*scale = 0 for Celcius to Farenheit
    scale = 1 for Farenheit to Celcius*/

    if (scale == 0) {
        // (X°C × 9 / 5) + 32 = Y°F

        converted_temp = (temperature * 1.8) + 32;
        return converted_temp;
    }

    else if (scale == 1) {
        //(Y°F − 32) × 5 / 9 = X°C

        converted_temp = (temperature - 32) * 5 / 9;
        return converted_temp;
    }

    else
        return -1;
}

int check_leap_year(int year) { //Check leap year function defination
    if (year < 0)
        return -1;

    else if (year % 100 == 0) {
        if (year % 400 == 0)
            return 1;
        else
            return 0;
    }

    else if (year % 4 == 0)
        return 1;

    else
        return 0;
}

int power_of_two_using_left_shit_operator(int exponent) { // Power of 2 using left
shit operator function defination

    if (exponent < 0)
        return -1;

    else
        return (2 << exponent - 1);
}
```

```
#include <math.h>
#include <stdio.h>
#include "Day2.h"

int sum_of_digits(int number) {      //Sum of digits function definition
        int count=0,sum=0,remainder;

        if (number < 0)
                return -1;

        while (number != 0) {
                remainder = number % 10;
                sum += remainder;
                count++;
                number /= 10;
        }

        return sum;
}

int reverse_number(int number) {      //Reverse number function definition
        int count = 0, num, remainder;

        num = number;
        while (num != 0)
        {
                num /= 10;
                count++;
        }

        if (count <= 1 || number < 0)
                return -1;

        while (number != 0)
        {
                remainder = number % 10;
                count--;
                num += remainder * pow(10, count);
                number /= 10;
        }

        return num;
}

int count_occurrences_of_digit(int digit, int number) { //Count the occurrences of
digit function definition


        int count = 0, remainder;
```

```
        while (number != 0) {

                remainder = number % 10;
                if (remainder == digit)
                        count++;
                number /= 10;
        }

        if (count > 0)
                return count;
        else
                return -1;
}

int check_palindrome(int number) { //Check palindrome of a number function
definition

        int rev_num;

        if (number < 0)
                return -1;

        rev_num = reverse_number(number);

        if (rev_num == number)
                return 1;
        else
                return 0;
}

int generate_n_prime_numbers(int n) { //Generate first 'N' prime numbers function
definition


        int count = 0,flag,input=2;

        while (count < n) {
                flag = 0;
                for (int i = 2; i <= input / 2; i++) {

                        if (input != i && input % i == 0) {

                                        flag = 1;
                                        break;
                        }
                }

                if (flag == 0) {
                        printf("%d\n", input);
                        count++;
```

```
                }

                input++;
        }
        return count;
}

int sum_of_series(int n) {
        int sum=0,i=1;

        while(i<=n){ //function definition To display and find the sum of the series
1+11+111+....111 up to n.
                sum = sum * 10 + i;
                i++;
        }
        printf("Sum is %d\n", sum);
        return sum;
}

int armstrong_number(int number) { //function that asks the user to enter a
numberand returns if it is Armstrong or not.

        int remainder, sum=0,og_num=number;

        if (number < 0)
                return -1;

        while (number != 0)
        {
                remainder = number % 10;
                sum += remainder * remainder * remainder;
                number /= 10;
        }

        if (og_num == sum)
                return 1;
        else
                return 0;
}

int amicable_numbers(int num1, int num2) {      //function to check that the input
pair of numbers is amicable

/*/Amicable numbers are found in pairs.A given pair of numbers
is Amicable if the sum of the proper divisors(not including itself) of one number is
equal to the other numberand vice – versa.*/

        int temp_num = 1,sum1=0,sum2=0;

        if (num1 < 0 || num2 < 0)
```

```
                return -1;

        while (temp_num <= num1 / 2) {

                if (num1 % temp_num == 0)
                        sum1 += temp_num;

                temp_num++;
        }

        temp_num = 1;

        while (temp_num <= num2 / 2) {

                if (num2 % temp_num == 0)
                        sum2 += temp_num;

                temp_num++;
        }

        if (sum1 == num2 && sum2 == num1)
                return 1;
        else
                return 0;
}

int calculator() {   // Function to read two integers& find their sum, difference&
product

        int num1, num2, op;

        printf("Enter the two numbers :\n");
        scanf_s(" %d %d", &num1, &num2);
        printf("Enter the option :\n Sum : 1\n Difference : 2\n product : 3\n");
        scanf_s(" %d", &op);

        switch (op) {
        case 1:
                return num1 + num2;

        case 2:
                return num1 - num2;

        case 3:
                return num1 * num2;

        default:
                return -1;
        }
}
```

```
double cube_volume_generator() { //Cube volume function

        double side,volume;

        printf("Enter the side length (in metre): \n");
        scanf_s(" %lf", &side);

        volume = side * side * side;
        printf("The volume of the cube is : %lf  metre cube\n", volume);
        return 0;
}

double cubiod_volume_generator() {   // Cubiod volume function

        double length, height, breadth, volume;

        printf("Enter the length of the cubiod (in metre):\n");
        scanf_s(" %lf", &length);

        printf("Enter the height of the cubiod (in metre):\n");
        scanf_s(" %lf", &height);

        printf("Enter the breadth of the cubiod (in metre):\n");
        scanf_s(" %lf", &breadth);

        volume = length * height * breadth;
        printf("The volume of the cubiod is : %lf metre cube\n", volume);
        return 0;
}

double sphere_volume_generator() { // sphere volume function

        double radius, volume;

        printf("Enter the radius of the sphere: (in metre)\n");
        scanf_s(" %lf", &radius);

        volume = 1.33333333333 * 3.14 * radius * radius * radius;
        printf("The volume of the sphere is : %lf metre cube\n", volume);
        return 0;
}

double cylinder_volume_generator() { // cylinder volume function

        double radius, height, volume;

        printf("Enter the radius of the cylinder: (in metre)\n");
        scanf_s(" %lf", &radius);
```

```
        printf("Enter the height of the cylinder (in metre):\n");
        scanf_s(" %lf", &height);

        volume = 3.14 * radius * radius * height;
        printf("The volume of the cylinder is : %lf metre cube\n", volume);
        return 0;
}

double cone_volume_generator() { // cone volume function

        double radius, height, volume;

        printf("Enter the radius of the cone: (in metre)\n");
        scanf_s(" %lf", &radius);

        printf("Enter the height of the cone (in metre):\n");
        scanf_s(" %lf", &height);

        volume = 3.14 * radius * radius * height/3;
        printf("The volume of the cone is : %lf metre cube\n", volume);
        return 0;
}

int electricity_bill_calculator(int units) { //Function to read no of unit conssumed
and print out total charge amount

        int total = 0;

        if (units < 0)
                return -1;

        else if (units <= 200) {
                total = 1 * units;
                return total;
        }

        else if (units <= 300) {
                total = 1 * 200 + 1.5 *(units-200);
                return total;
        }

        else {
                total = 1 * 200 + 1.5 * 100+ 2 * (units -300);
                return total;
        }

}
```

```
#include <math.h>
#include <stdio.h>
#include "Day3.h"

int convert_binary_to_decimal(int number) { // Function to convert a binary number
to decimal.

        int remainder, dec_num=0,i=0;

        while (number != 0) {

                remainder = number % 10;
                dec_num += remainder * pow(2, i);
                number /= 10;
                i++;
        }
        return dec_num;
}

int convert_decimal_to_binary(int number) { // Function to convert a  decimal number
to binary.

        int remainder, bin_num = 0, i = 0;

        while (number != 0) {

                remainder = number % 2;
                bin_num += remainder * pow(10, i);
                number /= 2;
                i++;
        }

        return bin_num;
}

int sequence_generator(int n) {
/*Function generating a sequence of numberssuch that every number in the
sequence is thesum of the previous three numbers.The first three numbers are 0, 0,
1. */
        // function returns no. of elements in sequence

        int sum, i=0, first_num=0, second_num=0, third_num=1;

        if (n <= 0)
                return -1;


        if (n == 3) {
                printf("\n%d,%d,%d", first_num, second_num, third_num);
                return n;
```

```
        }

        else if (n == 2) {
                printf("\n%d,%d", first_num, second_num);
                return n;
        }

        else if (n==1) {
            printf("\n%d", first_num);
                return n;
        }

        else {
                printf("\n%d,%d,%d", first_num, second_num, third_num);
                while ((i + 3) < n)
                {
                        sum = first_num + second_num + third_num;
                        first_num = second_num;
                        second_num = third_num;
                        third_num = sum;
                        printf(",%d", sum);
                        i++;
                }
                return i + 3;
        }
}

int pattern_generator(int row_no){
        /*Function to print the following sketch by taking in N as number of rows
                * * * *
                  * * *
                    * *
                      *     */

        int row=row_no, column,i=0;

        if (row <= 0)
                return -1;

        while (row > 0) {
                column = row;
                i = 0;
                while (column > 0) {

                        printf("*\t");
                        column--;
                }

                printf("\n");
```

```
                while (i + row <= row_no) {
                        printf("\t");
                        i++;
                }

                row--;
        }
        return row;
}

int magic_seven_numbers() {
//Function which will print two digit numbers whose sum of both digit is multiple of
seven.e.g. 16, 25, 34......

        int i=1, j,sum;

        while (i < 9)
        {
                j = 1;
                while (j < 9)
                {
                        sum = i + j;

                        if (sum % 7 == 0) {
                                sum = i * 10 + j;
                                printf("%d ", sum);
                        }
                        j++;
                }
                i++;
        }
        return 0;
}

int power_of_number_using_recursion(int base, int exponent) { //recursive function
for calculating power of a number

        if (base < 0 || exponent < 0)
                return -1;

        if (exponent == 0)
                return 1;
        else
                return base * power_of_number_using_recursion(base, exponent - 1);
}

float factorial_of_number_using_recursion(int number) {  // recursive function for
calculating factorial of a number

        if (number < 0)
```

```
                return -1;

        else if (number == 0)
                return 1;

        else
                return number * factorial_of_number_using_recursion(number - 1);
}

double series_generator(int x, int n) {
        // recursive calls to evaluate F(x) = x + (x*3 / 3!)+ (x*5 / 5!)+ (x*7 /
7!)+ …
        // "n" is no. of series elements
        static int odd_num=-1;

        if (n <= 0)
                return 0;

        else {
                odd_num += 2;
                return (x * odd_num) / factorial_of_number_using_recursion(odd_num)
+ series_generator(x, n-1);
        }

}

double series_generator_two(int x, int n) {
        // recursive calls to evaluate F(x) = x + (x^3 / 3!)+ (x^5 / 5!)+ (x^7 /
7!)+ …
        // "n" is no. of series elements
        static int odd_num = -1;

        if (n <= 0)
                return 0;

        else {
                odd_num += 2;
                return (pow(x, odd_num) /
factorial_of_number_using_recursion(odd_num)) + series_generator_two(x, n-1);
        }
}

int concatenate_two_integer() {
        //#define concatenate(x,y) x##y
        // The ##preprocessor transforms printf(“ % d”, concatenate(x, y)); into
printf(“ % d”, xy);

/* Allows tokens used as actual arguments to be concatenated to form other tokens.
It is often useful to merge two tokens into one while expanding macros.
This is called token pasting or token concatenation.The ‘##’ pre - processing
```

```
operator performs token pasting.
When a macro is expanded, the two tokens on either side of each ‘##’ operator are
combined into a single token,
which then replaces the ‘##’and the two original tokens in the macro expansion. */

        printf("%d",concatenate(7,4));
}

int macro_square(int number) {
        //function to find square of a number using macros
        return Macro_Square(number);
}

int math_function() {
        //Function to display the mathematical functions like square root, natural
log, log10x, power(x,n), Cos(x).
        int option;
        double x;

        printf("Select the mathematical function:\n Square root : 1\n Natural log :
2\n log10(x) : 3\n Power(x,n) : 4\n Cos(x) 'x in radians' : 5\n");
        scanf_s(" %d", &option);

        printf("Enter the value of x\n");
        scanf_s("%lf", &x);

        if (x < 0 || option < 0)
                return - 1;

        switch (option) {
        case 1:
                square_root(x);
                break;

        case 2:
                natural_log(x);
                break;

        case 3:
                log_to_base10(x);
                break;

        case 4:
                power(x);
                break;

        case 5:
                cos_function(x);
                break;
```

```
        default:
                return -2;
        }
}

int square_root(double x) {
        if (x == 0)
                return -1;
        printf("Square root of %0.1lf is %lf\n", x, sqrt(x));
        return 0;
}

int natural_log(double x) {
        if (x == 0)
                return -1;

        printf(" Natural log of %0.1lf is %lf\n", x, log(x));
        return 0;
}

int log_to_base10(double x) {
        if (x == 0)
                return -1;

        printf(" log10(%0.1lf) is %lf\n", x, log10(x));
        return 0;
}

int power(double x) {

        int n;

        if (x == 0)
                return -1;

        printf("Enter exponent value n : \n");
        scanf_s("%d", &n);
        printf(" %0.1lf to the power %d is %lf\n", x, n, pow(x, n));
        return 0;
}

int cos_function(double x) {

        printf("Value of Cos(%0.1lf) is %lf\n", x, cos(x));
        return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include "Day4.h"


 /* int array_size(int arr[]) {
        int i = 0;
        //int arr[] = a;
        //i = (*(&arr + 1) - arr);

        while (1)
        {
                if (arr[i] == NULL) {
                        printf("j %d  k %d\n", i, arr[i]);
                        break;
                }
                printf("i %d  k %d\n", i, arr[i]);
                i += 1;
        }
        return i;
}  */

    int* sort_array(int* arr, int size, int op, int print){             //Function
to sort an array
                int temp;
                //print_array(arr, size);

                if (op == 1) {                              //sorting array in
ascending order
                        for (int i = 0; i < size; i++) {
                                //printf("i %d\n", i);
                                for (int j = i + 1; j < size; j++) {

                                        if (arr[i] > arr[j]) {
                                                temp = arr[i];
                                                arr[i] = arr[j];
                                                arr[j] = temp;
                                        }
                                }

                        }
                }

                else {                                      //sorting array in
descending order
                        for (int i = 0; i < size; i++) {

                                for (int j = i + 1; j < size; j++) {

                                        if (arr[i] < arr[j]) {
```

```
                                        temp = arr[i];
                                        arr[i] = arr[j];
                                        arr[j] = temp;
                                }
                        }

                }
        }

        if (print == 1) {
                printf("\nRevised array after sorting are :\n");
                print_array(arr, size);
        }

    return arr;                     //returning the array is necessary for
binary search function
}

    int print_array(int* arr, int size) {
        int i = 0;
        if (size <= 0)
        {
                printf(" Error!! Invalid  Size %d ", size);
        }
        while (i<size)
        {
                printf("  %d\n", arr[i]);
                i++;
        }

        return 0;
    }

    int swap_numbers(int* a, int* b) {        //Swaping two numbers using call
by reference
        int temp;
        printf(" The numbers before swaping a = %d, b = %d\n", *a, *b);
        temp = *a;
        *a = *b;
        *b = temp;
        printf(" The numbers after swaping a = %d, b = %d\n", *a, *b);
        return 0;
    }

    int find_max_min(int array[], int size, int* max, int* min) {

        *max = *min = array[0];

                for (int j = 1; j < size; j++) {
```

```
                                if (*max < array[j])        //To find maximum
elements in a given array
                                        *max = array[j];

                                if (*min > array[j])       //To find minimum
elements in a given array
                                        *min = array[j];

                        }

                printf(" The minimum and maximum elements in a given array,\n max =
%d, min = %d\n", *max, *min);
                return 0;
        }

        int remove_duplicate_entries_array(int arr[], int size, int print) {

                int new_size = size;

                if (size <= 0)
                {
                        printf(" Error!! Invalid  Size %d ", size);
                }
                else {
                        for (int i = 0; i < new_size; i++) {

                                for (int j = i + 1; j < new_size; j++) { //for
traversing each entries in the array

                                        if (arr[i] == arr[j]) { // Checking for
duplicate entries in the array

                                                //printf("\ni%d j%d arr[i]%d
arr[j]%d new_size%d\n", i, j,  arr[i], arr[j], new_size);

                                                for (int k = j; k < new_size; k++) {
//
                                                        arr[k] = arr[k + 1];
                                                        //printf(" k%d  arr[k]%d
arr[k + 1]%d\n", k, arr[k], arr[k + 1]);
                                                }

                                                j = i;     //when there's two or
consecutive repeated elements eg : {1,1,1......}, j=i assures that the repeated
elements at 2+ position are also eliminated
                                                new_size--;       //Revising the
size of array after removing an element
                                        }

                                }
```

```
                        }
                }
                //printf("\nnew_size%d\n", new_size);

                if (print == 1) {
                        printf("\nRevised entires of the array without duplicate
entries are :\n");
                        print_array(arr, new_size);
                }

                return new_size;   //returning the size of the new array is
necessary for binary search function
        }

        int linear_search(int arr[], int size, int element) {

                if (size <= 0)
                {
                        printf(" Error!! Invalid  Size %d\n", size);
                        return -1;
                }

                for (int j = 1; j < size; j++) {

                        if (element == arr[j])        //To find element in a given
array
                                return 1;
                }

                return 0;
        }

        int binary_search_agorithm(int *sort_arr, int element, int start_position,
int end_position) { //Function to search for a given integer in an array using the
binary search technique

                int mid_value = start_position + (end_position - start_position) /
2;

                if (end_position < 0 || start_position > end_position) {
                        //printf("\nQ mid_value%d end_position%d start_position%d
\n", mid_value, end_position, start_position);
                        return 0;
                }

                else if (element == sort_arr[mid_value]) {
                        //printf("\nA mid_value%d end_position%d start_position%d
\n", mid_value, end_position, start_position);
                        return 1;
```

```
                }

                else if (element > sort_arr[mid_value]) {
                        start_position = mid_value + 1; // eliminating the values
that are in position before the mid value position i.e "element <
sort_arr[mid_value]"
                        //printf("\nG mid_value%d end_position%d start_position%d
\n", mid_value, end_position, start_position);
                        binary_search_agorithm(sort_arr, element, start_position,
end_position);
                }

                else {
                        end_position = mid_value - 1;  // eliminating the values
that are in position after the mid value position i.e "element <
sort_arr[mid_value]"
                        //printf("\nL mid_value%d end_position%d start_position%d
\n", mid_value, end_position, start_position);
                        binary_search_agorithm(sort_arr, element, start_position,
end_position);
                }

                return 0;

        }

        int binary_search(int arr[], int size, int element) {

                if (size <= 0)
                {
                        printf(" Error!! Invalid  Size %d\n", size);
                        return -1;
                }

                int* sort_arr = sort_array(arr, size,1,0);     // '1' to sort in
ascending order
                int new_size = remove_duplicate_entries_array(sort_arr, size,0);
                int start_position=0,end_position = new_size - 1;

                return binary_search_agorithm(sort_arr, element, start_position,
end_position);

        }

        int sum_of_product() {
                /* function to read list of 'n' integer and print sum of product of
consecutive numbers.
                   Ex: if n=7 and numbers are 4,5,2,5,6,4,7 then output is
4*5+5*2+2*5+5*6+6*4+4*7 = 122 */
```

```
		int size=0, sum=0,n,no;
		int* arr;
		printf("Enter the Number of integers to be entered\n");
		scanf_s("%d", &n);

		arr = (int*)malloc(n * sizeof(int));

		printf("Enter %d integers\n",n);
		while (size<n)
		{
			scanf_s("%d", &no);
			arr[size] = no;
			size++;
		}

		n -= 1;

		while (n > 0) {
			printf("(%d * %d)", arr[n - 1], arr[n]);
			sum += arr[n-1] * arr[n];
			n--;
			if (n> 0)
				printf(" + ");
		}

		printf(" = %d",sum);
		free(arr);
		return 0;
	}

	int length_of_string() {
		//function to read a string from the userand return the length of
string.
		int len = 0;
		char c[50];

		printf("Enter the string\n");
		gets(c);

		while (c[len] != '\0')
		{
			//printf("L %c\n", c[len]);
			len++;
		}
		printf("Length of the string is %d\n", len);
		return 0;
	}

	int worded_format_date() {
		/* Function to Input date, monthand year from the user, and using
```

```
switch case,
                display in worded format.e.g.input: d = 16, m = 7, y = 1992,Output:
16th July, 1992 */
                int year, date, month;
                char mon[12][15] =
{"January","Febuary","March","April","May","June","July","August","September","Octom
ber","November","December"};

                printf("Enter the year\n");
                scanf_s("%d", &year);

                printf("Enter the date\n");
                scanf_s("%d", &date);

                printf("Enter the month\n");
                scanf_s("%d", &month);

                if (year <= 0) {
                        printf(" Error!! Invalid  year no. %d ", year);
                        return 0;
                }

                if (date <= 0 || date > 31) {
                        printf(" Error!! Invalid  date  %d ", date);
                        return 0;
                }

                switch (month)
                {
                case 1:
                        switch (date)
                        {
                        case 1:
                                printf("%dst %s, %d\n",date,mon[0],year);
                                break;

                        case 21:
                                printf("%dst %s, %d\n", date, mon[0], year);
                                break;

                        case 2:
                                printf("%dnd %s, %d\n", date, mon[0], year);
                                break;

                        case 22:
                                printf("%dnd %s, %d\n", date, mon[0], year);
                                break;

                        case 3:
                                printf("%drd %s, %d\n", date, mon[0], year);
```

```
                        break;

                case 23:
                        printf("%drd %s, %d\n", date, mon[0], year);
                        break;

                default:
                        printf("%dth %s, %d\n", date, mon[0], year);
                        break;
                }

                break;


        case 2:
                switch (date)
                {
                case 1:
                        printf("%dst %s, %d\n", date, mon[1], year);
                        break;

                case 21:
                        printf("%dst %s, %d\n", date, mon[1], year);
                        break;

                case 2:
                        printf("%dnd %s, %d\n", date, mon[1], year);
                        break;

                case 22:
                        printf("%dnd %s, %d\n", date, mon[1], year);
                        break;

                case 3:
                        printf("%drd %s, %d\n", date, mon[1], year);
                        break;

                case 23:
                        printf("%drd %s, %d\n", date, mon[1], year);
                        break;

                default:
                        if (date > 29) {
                                printf(" Error!! Invalid date, %d doesnt
exist in %s", date,mon[1]);
                                return 0;
                        }
                        printf("%dth %s, %d\n", date, mon[1], year);
                        break;
                }
```

```
        break;

case 3:
        switch (date)
        {
        case 1:
                printf("%dst %s, %d\n", date, mon[2], year);
                break;

        case 21:
                printf("%dst %s, %d\n", date, mon[2], year);
                break;

        case 2:
                printf("%dnd %s, %d\n", date, mon[2], year);
                break;

        case 22:
                printf("%dnd %s, %d\n", date, mon[2], year);
                break;

        case 3:
                printf("%drd %s, %d\n", date, mon[2], year);
                break;

        case 23:
                printf("%drd %s, %d\n", date, mon[2], year);
                break;

        default:
                printf("%dth %s, %d\n", date, mon[2], year);
                break;
        }

        break;

case 4:
        switch (date)
        {
        case 1:
                printf("%dst %s, %d\n", date, mon[3], year);
                break;

        case 21:
                printf("%dst %s, %d\n", date, mon[3], year);
                break;

        case 2:
                printf("%dnd %s, %d\n", date, mon[3], year);
```

```
                        break;

                case 22:
                        printf("%dnd %s, %d\n", date, mon[3], year);
                        break;

                case 3:
                        printf("%drd %s, %d\n", date, mon[3], year);
                        break;

                case 23:
                        printf("%drd %s, %d\n", date, mon[3], year);
                        break;

                default:
                        if (date > 30) {
                                printf(" Error!! Invalid date, %d doesnt
exist in %s", date, mon[3]);
                                return 0;
                        }
                        printf("%dth %s, %d\n", date, mon[3], year);
                        break;
                }

                break;

        case 5:
                switch (date)
                {
                case 1:
                        printf("%dst %s, %d\n", date, mon[4], year);
                        break;

                case 21:
                        printf("%dst %s, %d\n", date, mon[4], year);
                        break;

                case 2:
                        printf("%dnd %s, %d\n", date, mon[4], year);
                        break;

                case 22:
                        printf("%dnd %s, %d\n", date, mon[4], year);
                        break;

                case 3:
                        printf("%drd %s, %d\n", date, mon[4], year);
                        break;

                case 23:
```

```
                        printf("%drd %s, %d\n", date, mon[4], year);
                        break;

                default:
                        printf("%dth %s, %d\n", date, mon[4], year);
                        break;
                }

                break;

        case 6:
                switch (date)
                {
                case 1:
                        printf("%dst %s, %d\n", date, mon[5], year);
                        break;

                case 21:
                        printf("%dst %s, %d\n", date, mon[5], year);
                        break;

                case 2:
                        printf("%dnd %s, %d\n", date, mon[5], year);
                        break;

                case 22:
                        printf("%dnd %s, %d\n", date, mon[5], year);
                        break;

                case 3:
                        printf("%drd %s, %d\n", date, mon[5], year);
                        break;

                case 23:
                        printf("%drd %s, %d\n", date, mon[5], year);
                        break;

                default:
                        if (date > 30) {
                                printf(" Error!! Invalid date, %d doesnt
exist in %s", date, mon[5]);
                                return 0;
                        }
                        printf("%dth %s, %d\n", date, mon[5], year);
                        break;
                }

                break;

        case 7:
```

```
            switch (date)
            {
            case 1:
                    printf("%dst %s, %d\n", date, mon[6], year);
                    break;

            case 21:
                    printf("%dst %s, %d\n", date, mon[6], year);
                    break;

            case 2:
                    printf("%dnd %s, %d\n", date, mon[6], year);
                    break;

            case 22:
                    printf("%dnd %s, %d\n", date, mon[6], year);
                    break;

            case 3:
                    printf("%drd %s, %d\n", date, mon[6], year);
                    break;

            case 23:
                    printf("%drd %s, %d\n", date, mon[6], year);
                    break;

            default:
                    printf("%dth %s, %d\n", date, mon[6], year);
                    break;
            }

            break;

    case 8:
            switch (date)
            {
            case 1:
                    printf("%dst %s, %d\n", date, mon[7], year);
                    break;

            case 21:
                    printf("%dst %s, %d\n", date, mon[7], year);
                    break;

            case 2:
                    printf("%dnd %s, %d\n", date, mon[7], year);
                    break;

            case 22:
                    printf("%dnd %s, %d\n", date, mon[7], year);
```

```
                        break;

                case 3:
                        printf("%drd %s, %d\n", date, mon[7], year);
                        break;

                case 23:
                        printf("%drd %s, %d\n", date, mon[7], year);
                        break;

                default:
                        printf("%dth %s, %d\n", date, mon[7], year);
                        break;
                }

                break;

        case 9:
                switch (date)
                {
                case 1:
                        printf("%dst %s, %d\n", date, mon[8], year);
                        break;

                case 21:
                        printf("%dst %s, %d\n", date, mon[8], year);
                        break;

                case 2:
                        printf("%dnd %s, %d\n", date, mon[8], year);
                        break;

                case 22:
                        printf("%dnd %s, %d\n", date, mon[8], year);
                        break;

                case 3:
                        printf("%drd %s, %d\n", date, mon[8], year);
                        break;

                case 23:
                        printf("%drd %s, %d\n", date, mon[8], year);
                        break;

                default:
                        if (date > 30) {
                                printf(" Error!! Invalid date, %d doesnt
exist in %s", date, mon[8]);
                                return 0;
                        }
```

```
                printf("%dth %s, %d\n", date, mon[8], year);
                break;
        }

        break;

case 10:
        switch (date)
        {
        case 1:
                printf("%dst %s, %d\n", date, mon[9], year);
                break;

        case 21:
                printf("%dst %s, %d\n", date, mon[9], year);
                break;

        case 2:
                printf("%dnd %s, %d\n", date, mon[9], year);
                break;

        case 22:
                printf("%dnd %s, %d\n", date, mon[9], year);
                break;

        case 3:
                printf("%drd %s, %d\n", date, mon[9], year);
                break;

        case 23:
                printf("%drd %s, %d\n", date, mon[9], year);
                break;

        default:
                printf("%dth %s, %d\n", date, mon[9], year);
                break;
        }

        break;

case 11:
        switch (date)
        {
        case 1:
                printf("%dst %s, %d\n", date, mon[10], year);
                break;

        case 21:
                printf("%dst %s, %d\n", date, mon[10], year);
                break;
```

```
                        case 2:
                                printf("%dnd %s, %d\n", date, mon[10], year);
                                break;

                        case 22:
                                printf("%dnd %s, %d\n", date, mon[10], year);
                                break;

                        case 3:
                                printf("%drd %s, %d\n", date, mon[10], year);
                                break;

                        case 23:
                                printf("%drd %s, %d\n", date, mon[10], year);
                                break;

                        default:
                                if (date > 30) {
                                        printf(" Error!! Invalid date, %d doesnt
exist in %s", date, mon[10]);
                                        return 0;
                                }
                                printf("%dth %s, %d\n", date, mon[10], year);
                                break;
                        }

                        break;

                case 12:
                        switch (date)
                        {
                        case 1:
                                printf("%dst %s, %d\n", date, mon[11], year);
                                break;

                        case 21:
                                printf("%dst %s, %d\n", date, mon[11], year);
                                break;

                        case 2:
                                printf("%dnd %s, %d\n", date, mon[11], year);
                                break;

                        case 22:
                                printf("%dnd %s, %d\n", date, mon[11], year);
                                break;

                        case 3:
                                printf("%drd %s, %d\n", date, mon[11], year);
```

```
                        break;

                case 23:
                        printf("%drd %s, %d\n", date, mon[11], year);
                        break;

                default:
                        printf("%dth %s, %d\n", date, mon[11], year);
                        break;
                }

                break;

        default:
                printf(" Error!! Invalid  month no. %d ", month);
                break;
        }

        return 0;
}

/* int lower_case_to_upper_case() {

        int len = 0;
        char character[50];

        printf("Enter the string\n");
        gets(character);

        while (character[len] != '\0')
        {
                len++;
        }
        //printf("len%d", len);

        for(int i=0; character[i] = '\0'; i++)
        {
                printf("\nReversed  :\n");
                //printf("%s :\n", character[i]);
                if(character[i]>= 'a' && character[i] <= 'z')
                character[i] -= 32;
        }
        character[len] = '\0';
        printf("\nReversed String :\n");
        printf("%s", character);
        //return character;
} */

int lower_case_to_upper_case(char low_string[]) {
        //Function to convert all lower - case characters into their upper -
```

```
case equivalents.
                char c[50];
                int len = 0;
                //len = strlen(low_string);

                while (low_string[len] != '\0') {

                        if (low_string[len] >= 'a' && low_string[len] <= 'z')
                        c[len] = low_string[len] - 32;
                        //printf("%c", c[len]);
                        len++;
                }
                c[len] = '\0';
                printf("%s\n", c);
                return 0;
        }

        int reverse_string() {
                //function to read a string from the userand reverse the strin
                int i = 0,len=0;
                char reverse[50],straight[50];

                printf("Enter the string\n");
                gets(straight);

                while (straight[len] != '\0')
                {
                        len++;
                }

                while (len >0 ) {
                        reverse[i] = straight[len-1];
                        //printf("f%d %c2 %c1 \n", len, reverse[i],
straight[len-1]);
                        i++;
                        len--;
                }
                reverse[i] = '\0';

                printf("\nReversed String :\n");
                printf("%s", reverse);
                return 0;
        }

        int matrix_5_cross_5()
        {
                //int **matrix;
                int row_size = 3, column_size = 3;

                int** matrix = construct_2Dmatrix(row_size, column_size);
```

```
        print_2Dmatrix(matrix, row_size, column_size);
        return 0;
}

int** construct_2Dmatrix(int row_size, int column_size)
{
        int** arr;
        int n;
        arr = (int**)malloc(row_size * sizeof(int*));

        for (int i = 0; i < row_size; i++)
        {
                arr[i] = (int*)malloc(column_size * sizeof(int));
        }

        for (int i = 0; i < row_size; i++)
        {
                for (int j = 0; j < column_size; j++)
                {
                        printf("matrix[%d][%d] = ",i,j);
                        scanf_s("%d", &n);
                        arr[i][j] = n;
                        printf("\n");
                }

        }
        return arr;
}

int print_2Dmatrix(int** arr, int row_size, int column_size)
{
        printf("matrix[%d][%d] = \n", row_size, column_size);

        for (int i = 0; i < row_size; i++)
        {
                for (int j = 0; j < column_size; j++)
                {
                        printf("%d  ",arr[i][j]);
                }
                printf("\n");
        }
        return 0;
}

int add_two_matrix()
{  //Declaration of function to add 2 matrixes

        int** sum;
        int row_size1=2,column_size1=2, row_size2,column_size2;
```

```
		row_size2 = row_size1;
		column_size2 = column_size1;
		int arr1[2][2] = { {9,4},{2,5} };
		int arr2[2][2] = { {1,1},{1,1} };
		if (row_size1 != row_size2 || column_size1 != column_size2) {

			printf("Error!!! For Matrix addition, matrices must have the
same dimensions\n");
			return -1;
		}



		sum = (int**)malloc(row_size1 * sizeof(int*));

		for (int i = 0; i < row_size1; i++)
		{
			sum[i] = (int*)malloc(column_size1 * sizeof(int));
		}

		for (int i = 0; i < row_size1; i++)
		{
			for (int j = 0; j < column_size1; j++)
			{
				sum[i][j] = arr1[i][j] + arr2[i][j];
			}
		}

		printf("The sum of the above 2 matrices :\n");
		print_2Dmatrix(sum, row_size1, column_size1);
		return 0;
	}

	/*
	int check_whether_two_matrixes_are_same(int** arr1, int row_size1, int
column_size1, int** arr2, int row_size2, int column_size2)
	{
		//int** array1;
		//int** array2;

		if (row_size1 != row_size2 || column_size1 != column_size2) {

			return 0;
		}

		for (int i = 0; i < row_size1; i++)
		{
			for (int j = 0; j < column_size1; j++)
			{
```

```
                        if (arr1[i][j] != arr2[i][j])
                                return 0;
                }

        }
        return 1;
}*/
int check_whether_two_matrixes_are_same()
{
        //function to check whether 2 matrixes are same.
        //int** array1;
        //int** array2;
        int row_size1 = 2, column_size1 = 3, row_size2, column_size2;

        row_size2 = row_size1;
        column_size2 = column_size1;

        int arr1[2][3] = { 0, 1 ,2 ,3 ,4 , 5 };
        int arr2[2][3] = { 1,1,1,1,1,1 };
        if (row_size1 != row_size2 || column_size1 != column_size2) {

                return 0;
        }

        for (int i = 0; i < row_size1; i++)
        {
                for (int j = 0; j < column_size1; j++)
                {
                        if (arr1[i][j] != arr2[i][j])
                                return 0;
                }

        }
        return 1;
}

int sparse_matrix()
{
/* function to check if given matrix is a sparse matrix.If the number of
zeros in a matrix exceeds(n * m) / 2,
        where n, m is the dimension of the matrix, matrix is sparse matrix.
        Sparse matrix has more zero elements than nonzero elements.*/

        int row_size1, column_size1,count=0, sparse_num;
        int** array;

        printf("Enter row size\n");
        scanf_s("%d", &row_size1);

        printf("Enter column size\n");
```

```
        scanf_s("%d", &column_size1);

        array = construct_2Dmatrix(row_size1, column_size1);

        print_2Dmatrix(array, row_size1, column_size1);

        for (int i = 0; i < row_size1; i++)
        {
                for (int j = 0; j < column_size1; j++)
                {
                        if (array[i][j] == 0)
                                count++;
                }

        }

        sparse_num = (row_size1 * column_size1) / 2;

        if(count > sparse_num)
                printf("It's sparse matrix\n");
        else
                printf("It's not sparse matrix\n");

        return 0;
}

int set_union(int arr1[], int size1, int arr2[], int size2)
{
        //Function to input and prints results of A union B

        int* union_set;
        int union_size,flag=0,count=0;

        union_size = size1 + size2;

        union_set = (int*)malloc(union_size * sizeof(int));

        for (int i = 0; i < size1; i++)
        {
                union_set[i] = arr1[i];
        }

        for (int i = 0; i < size2; i++)
        {
                for (int j = 0; j < size2; j++) {
                        if (arr2[i] == union_set[j]) {
                                flag = 1;
                                break;
                        }
                }
```

```
                        if (flag == 0) {
                                union_set[size1 + count] = arr2[i];
                                count++;
                        }
                        flag = 0;
                }

                printf(" Union set :\n");
                print_array(union_set, size1+count);
                free(union_set);
                *union_set = NULL;
                return 0;
        }

        int set_intersection(int arr1[], int size1, int arr2[], int size2)
        {
                int* intersection_set;
                int intersection_size, flag = 0, count = 0;

                if (size1 > size2)
                        intersection_size = size2;
                else
                        intersection_size = size1;

                intersection_set = (int*)malloc(intersection_size * sizeof(int));

                for (int i = 0; i < size1; i++)
                {
                        for (int j = 0; j < size2; j++) {

                                if (arr1[i] == arr2[j]) {
                                        //printf("arr1[i]%d arr2[j]%d\n", arr1[i],
arr2[j]);
                                        flag = 1;
                                        break;
                                }
                        }
                        if (flag == 1) {
                                intersection_set[count] = arr1[i];
                                count++;
                                flag = 0;
                        }

                }

                printf(" Intersection set :\n");
                print_array(intersection_set, count);
                free(intersection_set);
                *intersection_set = NULL;
                return 0;
```

```
	}

	int set_difference(int arr1[], int size1, int arr2[], int size2)
	{

		int* difference_set1, *difference_set2;
		int difference_size, flag = 0, count = 0;

		//Function to input and prints results of A-B and B-A
		difference_size = size1;

		difference_set1 = (int*)malloc(difference_size * sizeof(int));

		for (int i = 0; i < size1; i++)
		{
			for (int j = 0; j < size2; j++) {

				if (arr1[i] == arr2[j]) {
					//printf("arr1[i]%d arr2[j]%d\n", arr1[i],
arr2[j]);
					flag = 1;
					break;
				}
			}
			if (flag == 0) {
				difference_set1[count] = arr1[i];
				count++;
			}
			flag = 0;
		}

		printf(" Difference set Set1-Set2 :\n");
		print_array(difference_set1, count);
		free(difference_set1);
		*difference_set1 = NULL;

		//Function to input and prints results of B-A
		flag = 0;
		count = 0;
		difference_size = size2;

		difference_set2 = (int*)malloc(difference_size * sizeof(int));

		for (int i = 0; i < size2; i++)
		{
			for (int j = 0; j < size1; j++) {

				if (arr2[i] == arr1[j]) {
					//printf("arr1[i]%d arr2[j]%d\n", arr1[i],
arr2[j]);
```

```
                                flag = 1;
                                break;
                        }
                }
                if (flag == 0) {
                        difference_set2[count] = arr2[i];
                        count++;
                }
                flag = 0;
        }

        printf(" Difference set Set2-Set1 :\n");
        print_array(difference_set2, count);
        free(difference_set2);
        *difference_set2 = NULL;

        return 0;
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <string.h>
#include <stdio.h>
#include "Day6.h"


Student_Register student1_quickadd() {

	Student_Register student = { "Virat",1,20,97.7 };
	return student;
}

Student_Register student2_quickadd() {

	Student_Register student = { "Rohit",2,20,96.5 };
	return student;
}

Student_Register student3_quickadd() {

	Student_Register student = { "Rahul",3,18,95 };
	return student;
}

Student_Register student4_quickadd() {

	Student_Register student = { "Mayank",4,18,91.8 };
	return student;
}

Student_Register student5_quickadd() {

	Student_Register student = { "Smriti",5,17,93.6};
	return student;
}

Student_Register student6_quickadd() {

	Student_Register student = { "Shreyas",6,17,92.4};
	return student;
}

Student_Register student7_quickadd() {

	Student_Register student = { "Milthli",7,22,94.3};
	return student;
}

Student_Register student8_quickadd() {
```

```
        Student_Register student = { "Risbah",8,17,94.9};
        return student;
}

Student_Register student9_quickadd() {

        Student_Register student = { "Jasprit",9,18,96.96 };
        return student;
}

Student_Register student10_quickadd() {

        Student_Register student = { "Jullan",10,21,94.7};
        return student;
}

int print_student_register(Student_Register student)
{
        Student_Register std;

        strcpy(std.name, student.name);
        std.register_number = student.register_number;
        std.age = 10;
        std.percentage = student.percentage;
        printf("Student Name : %s\nRollNo. : %d\nAge : %d\nPercentge : %0.2f
%\n\n",std.name,std.register_number,std.age,std.percentage);

        return 0;
}

ComplexNumber complex_addition(ComplexNumber number1, ComplexNumber number2)
{
        ComplexNumber answer;

        answer.real_number = number1.real_number + number2.real_number;
        answer.imaginary_number = number1.imaginary_number +
number2.imaginary_number;

        return answer;
}

ComplexNumber complex_subtraction(ComplexNumber number1, ComplexNumber number2)
{
        ComplexNumber answer;

        answer.real_number = number1.real_number - number2.real_number;
        answer.imaginary_number = number1.imaginary_number -
number2.imaginary_number;

        return answer;
```

```
}

ComplexNumber complex_multiplication(ComplexNumber number1, ComplexNumber number2)
{
        ComplexNumber answer;

        answer.real_number = number1.real_number * number2.real_number;
        answer.imaginary_number = number1.imaginary_number *
number2.imaginary_number;

        return answer;
}

int print_complex_number(ComplexNumber number1)
{
        printf("%0.1f + %0.1fj\n", number1.real_number, number1.imaginary_number);
        return 0;
}

int billing()
{
        int ex=1,no_items,quantity;
        float price,total=0,grand_total;
        float *subtotal;
        GrandTotal *bill;
        printf("Enter no. of items\n");
        scanf("%d",&no_items);

        bill = (GrandTotal*)malloc(no_items * sizeof(GrandTotal));
        subtotal = (float*)malloc(no_items * sizeof(float));

        for (int i = 0; i < no_items; i++) {
                printf("\nEnter Item%d Price :",i+1);
                scanf_s("%f", &price);
                bill[i].item_price = price;

                printf("Enter Item%d Quantity :",i+1);
                scanf_s("%d", &quantity);
                bill[i].item_quantity = quantity;
        }

        for (int i = 0; i < no_items; i++) {
                //printf("total%d %0.2f,
bill[i].item_price%f,bill[i].item_quantity%d\n",i, total, bill[i].item_price,
bill[i].item_quantity);
                subtotal[i] = bill[i].item_price * bill[i].item_quantity;
        }

        for (int i = 0; i < no_items; i++) {
                //printf("total%d %0.2f,
```

```
bill[i].item_price%f,bill[i].item_quantity%d\n",i, total, bill[i].item_price,
bill[i].item_quantity);
                total += subtotal[i];
        }
        grand_total = discount_calculator(total);

        //printf("grand total %0.2f\n", total);

        print_bil(bill,subtotal,grand_total,total,no_items);
        return 0;
}

float discount_calculator(float total)
{
        //int grand_total;

        if (total < 1000) {
                total -= total * 0.05;
                return total;
        }
        else if(total<5000) {
                total -= total * 0.1;
                return total;
        }
        else{
                total -= total * 0.15;
                return total;
        }

}

int print_bil(GrandTotal *items, float subtotal[], float grand_total, float total,
int no_items)
{
        float discount;

        printf("\nItems\t\tPrice\t\tQuantity\t\tSubtotal\n");

        for (int i = 0; i < no_items; i++) {

                printf("Items %d\t\t%0.2f\t\t   %d\t\t\t %0.2f\n",i+1,
items[i].item_price, items[i].item_quantity,subtotal[i]);
        }


printf("\n---------------------------------------------------------------------\n");
        printf("\nTOTAL %0.2f\n", total);

        if (total < 1000) {
                discount = total * 0.05;
```

```
                printf("Discount 5'%' - %0.2f\n", discount);
        }
        else if (total < 5000) {
                discount = total * 0.10;
                printf("Discount 10'%' - %0.2f\n", discount);
        }
        else {
                discount = total * 0.15;
                printf("Discount 15'%' - %0.2f\n", discount);
        }


printf("\n-------------------------------------------------------------------------\n");
        printf("GRAND TOTAL %0.2f\n", grand_total);
        return 0;
}
```

```
#include <stdio.h>
#include "Day7.h"

int* construt_array(int array[], int size)
{
	int x;
	//int* array1 =array;

	printf("\nEnter the integers\n");

	for (int i = 0; i < size; i++)
	{
		scanf_s("%d", &x);
		*(array + i) = x;
	}

	//printf("Integers in the Array are:\n");
	//print_array(array1, size);

	return array;
}

int print_array(int* arr, int size) {
	int i = 0;
	if (size <= 0)
	{
		printf(" Error!! Invalid  Size %d ", size);
	}
	while (i < size)
	{
		printf("  %d\n", arr[i]);
		i++;
	}

	return 0;
}

int linear_search(int arr[], int size) {
	//function to search for a given integer in an array using the linear search
technique
	int element,flag=0;

	printf("\nEnter the element to be searched\n");
	scanf_s("%d", &element);

	if (size <= 0)
	{
		printf(" Error!! Invalid  Size %d\n", size);
		return -1;
	}
```

```
        for (int j = 0; j < size; j++) {

                if (element == arr[j]) {
                        printf("The Element %d is present in the array\n", element);
  //To find element in a given array
                        flag=1;
                        break;
                }

        }

        if(flag==0)
                printf("The Element %d is not present in the array\n", element);

        return 0;
}

Array_Min_Max find_max_min(int array[], int size) {
        //Function to find Max and Min element in an array


        Array_Min_Max element1;
        element1.max = element1.min = array[0];

        for (int j = 1; j < size; j++) {

                if (element1.max < array[j])        //To find maximum elements in a
given array
                        element1.max = array[j];

                if (element1.min  > array[j])       //To find minimum elements in a
given array
                        element1.min = array[j];

        }

        //printf(" The minimum and maximum elements in a given array,\n max = %d,
min = %d\n", *max, *min);
        return element1;
}

int sum_array(int array[], int size)
{
        //function to find sum of all the elements of array
        int sum=0;

        for (int i = 0; i < size; i++)
        {
                sum += array[i];
```

```
        }
        return sum;
}

Elements_Array sum_of_array_elements(int array[], int size)
{ //function to find the sum of even and odd elements of array
        int even_sum=0, odd_sum=0;
        Elements_Array arr1;

        for (int i = 0; i < size; i++)
        {
                if (array[i] % 2 == 0)
                        even_sum += array[i];
                else
                        odd_sum += array[i];
        }
        arr1.sum_of_even_elements = even_sum;
        arr1.sum_of_odd_elements = odd_sum;

        return arr1;
}

int array_palindrome(int array[], int size)
{ //function to Check whether the array in palindrome
        int flag = 1;

        for (int i = 0; i < size/2; i++)
        {
                if (array[i] != array[size-1-i]) {
                        flag = 0;
                        break;
                }
        }

        if (flag)
                printf("\nArray is a palindrome\n");
        else
                printf("\nArray is not a palindrome\n");

        return flag;
}

int* deallocate_memory(int* array)
{
        //function to deallocate the memory using free()

        free(array);
        *array = NULL;
        return array;
```

```
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#include "Day8.h"

Student_Register update_register()
{
        //function to enter the details of student
        char name[20]={0};
        int register_no, age, marks,name_size;

        Student_Register student;

        printf("\nEnter the name of student\n");
        scanf("%s", &name);
        name_size = strlen(name);
        strcpy(student.name, name);
        student.name[name_size] = '\0';


        printf("Enter Register number\n");
        scanf_s("%d", &register_no);
        student.register_number = register_no;

        printf("Enter age\n");
        scanf_s("%d", &age);
        student.age = age;

        printf("Enter Total marks out 600\n");
        scanf_s("%d", &marks);
        student.marks = marks;

        return student;
}

int print_student_register(Student_Register student)
{
        printf("\n\nStudent Details :-\nStudent Name : %s\nRegister Number : %d\nAge
: %d\nMarks : %d", student.name,student.register_number,student.age,student.marks);
        return 0;
}

Student_Register search_by_name(Student_Register* student,int size)
{
        //function to Search student by name
        int flag=-1;
        char name[20];

        printf("\n\nSearch student by name...\nEnter the name of the student\n");
        scanf("%s", &name);
```

```
        for (int i = 0; i < size; i++)
        {
                //printf("\nstudent[i].name %s %s\n", _strlwr(student[i].name),
_strlwr(name));

                if ( !(strcmp(_strlwr(student[i].name), _strlwr(name))) ){
                        print_student_register(student[i]);
                        flag = i;
                }
        }
        if (flag == -1) {
                printf("\nThere's no student with name %s in the register\n",name);
        }

        return student[flag];
}

Student_Register search_by_rollnum(Student_Register* student, int size)
{
        //function to Search student by roll no.
        int flag = -1, rno;

        printf("\n\nSearch student by register number...\nEnter the register number
of the student\n");
        scanf("%d", &rno);

        for (int i = 0; i < size; i++)
        {
                if (student[i].register_number == rno) {
                        print_student_register(student[i]);
                        flag = i;
                }
        }
        if (flag == -1) {
                printf("\nThere's no student with register number %d in the
register\n", rno);
        }

        return student[flag];

}

Student_Register details_by_starting_letter_of_name(Student_Register* student, int
size)
{
        //function to Display the details of the students whose name begins with
‘A’.
        int flag = -1;
        char letter;
```

```
        char name[20];

        printf("\n\nDetails of student with starting letter of name...\nEnter the
letter\n");
        scanf(" %c", &letter);

        for (int i = 0; i < size; i++)
        {
                strcpy(name, student[i].name);
                //printf("\nstudent[i].name %c %c\n", toupper(name[0]),
toupper(letter));

                if (toupper(name[0]) == toupper(letter)) {
                        print_student_register(student[i]);
                        flag = i;
                }
        }

        if (flag == -1) {
                printf("\nThere's no student with name with starting letter %c in
the register\n", letter);
        }

        return student[flag];
}

Student_Register merit_list(Student_Register* student, int size)
{
        // Function to Return student details who have scored highest marks.
        int highest_marks=0,n=0;

        for (int i = 0; i < size; i++) {

                if (student[i].marks > highest_marks) {
                        highest_marks = student[i].marks;
                        n = i;
                }
        }
        return student[n];
}

Student_Register* deallocate_memory(Student_Register* student)
{ //function to deallocate the memory using free()

        free(student);
        student = NULL;

        return student;
}
```

```
double weight_calculator(XY_Plane xy)
{// Declaration of function to calculate weight of the points.
        double weight= xy.x * xy.y;

        return weight;
}

XY_Plane add_coordinate(int i)
{// Declaration of function to enter the coordinates of the point.
        float x, y;

        XY_Plane xy;

        printf("\nEnter X%d coordinate : \n",i+1);
        scanf("%f", &x);
        xy.x = x;

        printf("\nEnter Y%d coordinate : \n",i+1);
        scanf("%f", &y);
        xy.y = y;

        return xy;
}

XY_Plane maximum_weight(XY_Plane* xy, int no_of_points)
{// Declaration of function to find the point with maximum weight.
        double max_weight = 0;
        int n=0;

        for (int i = 0; i < no_of_points; i++) {

                if (xy[i].weight > max_weight) {
                        max_weight = xy[i].weight;
                        n = i;
                }
        }
        return xy[n];
}

int vertical_line_counter(XY_Plane* xy, int no_of_points)
{// Declaration of function to count the number of vertical lines.
        int count = 0;
        //float k, h;

        for (int i = 0; i < no_of_points; i++) {
                //k = ;
```

```
                for (int j = i+1; j < no_of_points; j++) {
                        //h = xy[i].x;

                        if (xy[i].x == xy[j].x)
                                count++;
                }
        }

        return count;
}

int horizontal_line_counter(XY_Plane* xy, int no_of_points)
{// Declaration of function to count the number of horizontal lines.
        int count = 0;
        //float k, h;

        for (int i = 0; i < no_of_points; i++) {
                //k = ;
                for (int j = i + 1; j < no_of_points; j++) {
                        //h = xy[i].x;

                        if (xy[i].y == xy[j].y)
                                count++;
                }
        }

        return count;
}

int print_point(XY_Plane xy)
{
        printf("\nPoint(%0.2f,%0.2f)\nX coordinate : %0.2f\nY coordinate :
%0.2f\nWeight : %0.2f\n", xy.x,xy.y, xy.x,xy.y, xy.weight);
        return 0;
}

XY_Plane* deallocate_memory2(XY_Plane* xy)
{ // Declaration of function to deallocate the memory using free()
        free(xy);
        xy = NULL;

        return xy;
}

char* lower_case_to_upper_case(char low_string[]) {
        //Function to convert all lower - case characters into their upper - case
equivalents.
        char c[50];
        int len = 0;
        //len = strlen(low_string);
```

```
        while (low_string[len] != '\0') {

                if (low_string[len] >= 'a' && low_string[len] <= 'z')
                        c[len] = low_string[len] - 32;
                //printf("%c", c[len]);
                len++;
        }
        c[len] = '\0';
        //printf("%s\n", c);
        return c;
}
```

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#include "Day9.h"

int create_file() {

	FILE* fp;

	fp = fopen("ReadMe.txt", "w");

	fprintf(fp, "%s", "a simple program to display the contents of a file.");
	fclose(fp);

	return 0;
}

int display_file()
{
	// Function to display the contents of a file.
	FILE* fp1;

	fp1 = fopen("ReadMe.txt", "r");
	char ch = "";


	if (fp1 == NULL) {

		printf("Unable to open the file");
		return 0;
	}

	while (ch != EOF) {
		//printf("v\n");
		ch = fgetc(fp1);
		printf("%c", ch);
	}

	fclose(fp1);

	return 0;
}

int copy_file()
{// Function to copy the contents of one file to another

	FILE* fp2, *fp3;
	char ch ="";

	fp2 = fopen("CopyMe.txt", "w");
```

```
        fp3 = fopen("ReadMe.txt", "r");

        while (ch != EOF) {
                //printf("v\n");
                ch = fgetc(fp3);
                fputc(ch,fp2);
                printf("%c", ch);
        }
        fclose(fp2);
        fclose(fp3);

        return 0;
}

int counter()
{// Function to count number of characters, spaces tabs and lines in a file.
        FILE* fp4;
        char ch="";
        int no_space = 0, no_tabs=0, no_line = 1, no_char = 0;

        fp4 = fopen("ReadMe.txt", "r");

        if (fp4 == NULL) {

                printf("Unable to the file");
                return 0;
        }

        while (1) {
                ch = getc(fp4);

                if (ch == EOF)
                        break;

                if (ch == '\n')
                        no_line++;

                if (ch == '\t')
                        no_tabs++;

                if (ch == ' ')
                        no_space++;

                no_char++;
        }

        printf("No. of Character = %d\nNo. of lines = %d\nNo. of tabs = %d\nNo. of
space = %d\n",no_char,no_line,no_tabs,no_space);

        fclose(fp4);
```

```
        return 0;
}

int odd_or_even()
{                   //Read a file which contains one number per line.
                    //Check the number is odd or even and write to corresponding
odd.txt or even.txt accordingly.
                    //Note: Numbers may not be single digit numbers.
        FILE* fp5, * fp6, * fp7;

        int ch=0;

        fp5 = fopen("Numbers.txt","r");

        if (fp5 == NULL) {

                printf("Unable to the file");
                return 0;
        }

        while (!feof(fp5)) {
                fscanf(fp5, "%d", &ch);

                if (ch % 2 == 0) {
                        //printf("%d", ch);
                        fp6 = fopen("even.txt", "a");
                        fprintf(fp6, "%d\n", ch);
                        fclose(fp6);
                }

                else {
                        //printf("o%d", ch);
                        fp7 = fopen("odd.txt", "a");
                        fprintf(fp7, "%d\n", ch);
                        fclose(fp7);
                }
        }

        fclose(fp5);
        return 0;
}

int create_telephone_directory()
{
        //Function to create a directory

        FILE* fp8;

        fp8 = fopen("telephone_directory.txt", "w");   //Opening the directory in
```

```
"write mode"

        if (fp8 == NULL) { //If "fopen" function  fails to open a file, -1 is
returned.

                printf("Unable to create the directory");
                return -1;
        }

        fclose(fp8);

        return 0;
}

int add_contacts()
{
        //Function to add entries to the directory created using
"create_telephone_directory()" function
        FILE *fp9;
        char address[100], name[30];
        int phone_number;

        fp9 = fopen("telephone_dictionary.txt", "a");       //Opening the directory
in "append mode"


        printf("Enter the phone number :\n");
        scanf("%d", &phone_number);                 //Asking the user to enter the
number
        fprintf(fp9, "%d\n", phone_number);         //printing the number the user
has entered to the file (dictionary)

        printf("Enter the name :\n");
        scanf("%s", &name);                         //Asking the user to enter the
name
        fprintf(fp9,"%s\n",name);                   //printing the name the user has
entered to the file (dictionary)

        printf("Enter the address :\n");
        scanf("%s", &address);                     //Asking the user to enter the
address
        fprintf(fp9, "%s\n", address);             //printing the address the user
has entered to the file (dictionary)

        fprintf(fp9,"\n");                  //Moving the cursor to the next line so
that there's gap of 1 line between the entries of 2 users

        fclose(fp9);

        return 0;
```

```
}

/* int number_finder() {

        FILE* fp10;
        char num[100];
        char file_num[15]="";
        int count,line_num=0,flag=0;

        printf("Enter the number to be searched\n");
        scanf("%s", &file_num);

        fp10= fopen("telephone_dictionary.txt", "r");

        if (fp10 == NULL) {
                printf("Unable to open the directory");
                return -1;
        }

        while (!feof(fp10)) {    //we search till End Of File, feof returns non zero
number when end of file is reached.

                fgets(num,11, fp10); //we read a line from the file. '11' because we
enter 10 (because phone num contains 10 numbers) it reads 9 numbers
                                     //maybe one extra character for '\0' (null
character )[don't know not sure :( ]so 11.

                 /* printf("num %s l%d\n", num, line_num);

                if (!(strcmp(num, file_num))) {

                        flag = 1;    //If the 'number' read from the file matches
the 'number to be searched, we maake flag=1 and proceed to print the details

                        printf("\nNumber %s found in the directory\nDetails :\n",
file_num);
                        count = 3;  //counter to print next 2 lines i.e. the name
and the address.
                                    //count = 3 because for the 1st iteration fgets
reads blank line.[again don't know y :( ].
                                    //hence we intialze count =3 (one number extra
than required).

                        while (count > 0) {
                                fgets(num, 100, fp10); // '100' because address max
size is set 100, and we are reading address also.

                                if (count == 2)
                                        printf("Name : %s\n", num);
```

```
                                if(count==1)
                                        printf("Address : %s\n", num);

                                count--;
                        }

                        fclose(fp10);   //After displaying the details, we close the
file and return the line number the "phone number" was found in the directory
                        return line_num;

                }
                else {           //If there's no match, we proceed to read the next
line and increment the line counter(i.e line_num)
                        line_num++;
                }
        }



        if (!flag) { //if flag is '0' it means that Number doesn't exist in the
directory, hence we close the file and return 0

                printf("\nNumber %s doesn't exist in the directory\nDetails :",
file_num);
                fclose(fp10);
                return 0;
        }



} */

/* int edit_telephone_directory()
{
        FILE* fp11, *fp12;

        char data[100];
        //char new_num[10];
        int line_num = 0;

        printf("Enter the number to be edited\n");
        //scanf("%s", &new_num);

        fp11 = fopen("telephone_dictionary.txt", "r");

        if (fp11 == NULL) {
                printf("Unable to open the directory");
                return -1;
        }
```

```
        fp12 = fopen("temperory_dictionary.txt", "a");


        while (line_num < 5) {

                fgets(data, 100, fp11);
                fprintf(fp12, "%s", data);

                fgets(data, 30, fp11);
                fprintf(fp12, "%s", data);

                fgets(data, 100, fp11);
                fprintf(fp12, "%s\n", data);

                printf("line : %d", line_num);
                line_num++;
        }

        fclose(fp11);
        fclose(fp12);

        return 0;
} */

int update_directory_entry(Line_Details details) {

        FILE* fp11;

        char data[] = "";
        //char new_num[10];

        printf("Enter the updated number \n");
        scanf("%s", &data);                  //Asking the user to enter the new
number to be replaced with the old number

        fp11 = fopen("telephone_dictionary.txt", "r+"); //Opening the directory in
"read & write mode"

        if (fp11 == NULL) { //If "fopen" function  fails to open a file, -1 is
returned.
                printf("Unable to open the directory");
                return -1;
        }


        fseek(fp11, details.char_position, SEEK_SET);    //setting the cursor
position to the line where the searched number was found

        //fputs("9111122332", fp11);  //overwriting old number with new one.
```

```
        fputs(data, fp11);  //where 'data' is declared as char data="", and taken
from the user;
        //fprintf(fp11, "%s", data);

        fclose(fp11);

        //printf("lb%d\nlp%d", l1.line_number, l1.char_position);
        return 0;

}


int delete_directory_entry(Line_Details details) {

        FILE* fp12;
        int count = 0;

        fp12 = fopen("temperory_dictionary.txt", "r+"); //Opening the directory in
"read & write mode"

        if (fp12 == NULL) { //If "fopen" function  fails to open a file, -1 is
returned.
                printf("Unable to open the directory\n");
                return -1;
        }

        fseek(fp12, details.char_position, SEEK_SET);    //setting the cursor
position to the line where the searched number was found

        while (count < 3) { //Itirating 3 times so that data in consecutive 3
lines(Phone number,name,address) gets overwriiten with white space.
                fputs("            ", fp12); //Overwriting the existing data gets
overwritten with 12 white spaces
                count++;
        }

        printf("The above entry is successfully deleted from the directory\n");
        fclose(fp12);

        return 0;

}

Line_Details directory_search()
{
        FILE* fp13;
        char num[100];
        char file_num[15] = "";
        Line_Details l1;     //declaring a structure to store line number and
character position.
```

```
        int count, flag = 0;

        printf("Enter the number to be searched\n");  //User enters the number to be
searched
        scanf("%s", &file_num);

        fp13 = fopen("telephone_dictionary.txt", "r"); //Opening the directory in
"read mode"

        l1.line_number = -1;
        l1.char_position = -1;

        if (fp13 == NULL) {   //If "fopen" function  fails to open a file,Structure
'l1' with  line_number & char_position=-1 is returned
                printf("Unable to open the directory\n");
                return l1;
        }

        while (!feof(fp13)) {    //searches till End Of File, feof returns non zero
number when end of file is reached.

                l1.char_position = ftell(fp13); //ftell returns position of file
pointer in the file with respect to starting of the file

                fgets(num, 11, fp13); //reads a line from the file. '11' because we
enter 10 (because phone num contains 10 numbers) it reads 9 numbers
                                                        //maybe one extra character
for '\0' (null character )[don't know not sure :( ]so 11.



                 /* printf("num %s l%d\n", num, line_num); */

                if (!(strcmp(num, file_num))) {

                        flag = 1;    //If the 'number' read from the file matches
the 'number to be searched, flag is made 1 (i.e. flag=1) and proceed to print the
details

                        printf("\nNumber %s found in the directory\nDetails :\n",
file_num);
                        count = 3;  //counter to print next 2 lines i.e. the name
and the address.
                                                //count = 3 because for the 1st
iteration fgets reads blank line.[again don't know y :( ].
                                                //hence intialze count =3 (one
number extra than required).

                        while (count > 0) {
                                fgets(num, 100, fp13); // Since address also been
```

```
read by the same "fgets", offset value is made '100' (because address max size is
declared as 100).

                                if (count == 2)
                                        printf("Name : %s\n", num);

                                if (count == 1)
                                        printf("Address : %s\n", num);

                                count--;
                        }

                        /*printf("lwe%d", l1.char_position); */

                        fclose(fp13); //After displaying the details, we close the
file and
                        return l1;  //return the structure l1 which contains line
number & char_position of the "phone number" found in the directory

                }
                else {      //If there's no match, proceed to read the next line and
increment the line counter(i.e line_number)
                        l1.line_number++;
                }
        }


        if (!flag) { //if flag is '0' it means that Number doesn't exist in the
directory, hence close the file and
                                 //return the structure l1 with line number &
char_position equal to "0"

                printf("\nNumber %s doesn't exist in the directory\n", file_num);
                l1.char_position = 0;
                l1.line_number = 0;
                fclose(fp13);
                return l1;
        }
}



int create_telephone_directory_binary()
{

        //Function to create a directory

        FILE* fp14;

        fp14 = fopen("telephone_directory_binary.txt", "wb");   //Opening the
```

```
directory in "write mode"

        if (fp14 == NULL) { //If "fopen" function  fails to open a file, -1 is
returned.

                printf("Unable to create the directory");
                return -1;
        }

        fclose(fp14);

        return 0;
}

int add_contacts_binary()
{
        //Function to add entries to the directory created using
"create_telephone_directory_binary()" function
        FILE* fp15;
        char address[100], name[30];
        int phone_number;

        fp15 = fopen("telephone_directory_binary.txt", "ab");       //Opening the
directory in "append mode"


        printf("Enter the phone number :\n");
        scanf("%d", &phone_number);                //Asking the user to enter the
number
        fwrite(&phone_number, sizeof(phone_number), 1, fp15);   //printing the
number the user has entered to the file (dictionary)

        printf("Enter the name :\n");
        scanf("%s", &name);                        //Asking the user to enter the
name
        fwrite(&name, sizeof(name), 1, fp15);                  //printing the name
the user has entered to the file (dictionary)

        printf("Enter the address :\n");
        scanf("%s", &address);                    //Asking the user to enter the
address
        fwrite(&address, sizeof(address), 1, fp15);            //printing the
address the user has entered to the file (dictionary)

        //fprintf(fp9, "\n");                 //Moving the cursor to the next line
so that there's gap of 1 line between the entries of 2 users

        fclose(fp15);

        return 0;
```

```
}
```