1. Write a C program to multiply two matrices using dynamic memory allocation. Each two-dimensional array should be processed as array of pointers to a set of 1-dimensional integer arrays. Read, access and display the matrix elements using pointers instead of subscript notation. Use three functions i) To read input matrix ii) To compute the product and iii) To display the resultant matrix.

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

void read(int **matrix, int size)
{
    printf("Enter elements of the matrix\n");
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            scanf("%d", (*(matrix + i) + j));
        }
    }
}

void multi(int **a, int **b, int **result, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            *(*(result + i) + j) = 0;
            for (int k = 0; k < size; k++)
            {
                *(*(result + i) + j) += *(*(a + i) + k) * *(*(b + k) + j);
            }
        }
    }
}

void display(int **matrix, int size)
{
    printf("Matrix:\n", size);
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d ", *(*(matrix + i) + j));
        }
        printf("\n");
    }
}

int main()
{
    int size;
    printf("Enter matrix size: \n");
    scanf("%d", &size);

    int **a = (int **)malloc(size * sizeof(int *));
    int **b = (int **)malloc(size * sizeof(int *));
    int **result = (int **)malloc(size * sizeof(int *));

    for (int i = 0; i < size; i++)
        *(a + i) = (int *)malloc(size * sizeof(int));
    for (int i = 0; i < size; i++)
        *(b + i) = (int *)malloc(size * sizeof(int));
    for (int i = 0; i < size; i++)
        *(result + i) = (int *)malloc(size * sizeof(int));

    read(a, size);
    read(b, size);
    multi(a, b, result, size);

    printf("Resultant Matrix:\n");
    display(result, size);

    for (int i = 0; i < size; i++)
        free(*(a + i));
    for (int i = 0; i < size; i++)
        free(*(b + i));
    for (int i = 0; i < size; i++)
        free(*(result + i));
    free(a);
    free(b);
    free(result);
    return 0;
}
```

2. Write a C program to hold two integer pointers as structure members. Allocate space for the structure and its data members during runtime. Get one array as input. In the second array copy the elements of the first array and replace the odd positioned elements by the product of its adjacent elements. Access the array elements and structures using pointers instead of subscript notation.

First Array (Input)

1	2	3	4	5	6

Second Array (Input)

1	3	3	15	5	6

```c
#include <stdlib.h>

struct Arrays {
    int *arr1;
    int *arr2;
};

void replaceOddPositions(int *arr1, int *arr2, int n) {
    for(int i = 0; i < n; i++) {
      *(arr2 + i) = *(arr1 + i);
        if(i % 2 != 0 && i > 0 && i < n - 1) {
       *(arr2 + i) = *(arr1 + (i - 1)) * *(arr1 + (i + 1));
        }
    }
}

int main() {
    struct Arrays arrays;
    int n;

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

    arrays.arr1 = (int *)malloc(n * sizeof(int));
    arrays.arr2 = (int *)malloc(n * sizeof(int));

    if(arrays.arr1 == NULL || arrays.arr2 == NULL) {
        printf("Memory allocation failed\n");
        return -1;
    }

    printf("Enter the elements of the first array:\n");
    for(int i = 0; i < n; i++) {
        scanf("%d", arrays.arr1 + i);
    }

    replaceOddPositions(arrays.arr1, arrays.arr2, n);

    printf("Second Array (Output):\n");
    for(int i = 0; i < n; i++) {
        printf("%d ", *(arrays.arr2 + i));
    }

    free(arrays.arr1);
    free(arrays.arr2);

    return 0;
}
```

3. Create a Structure called BankMgmt with AccNumber, CustName, AvlBalance, AccType as members. Implement a Bank management Application as menu driven program for the above said Structure scenario.

   Menu Option:

   1. Withdrawal 2. Deposit 3. Display Balance 4. Exit

   If option

   1 is chosen- Amount can be withdrawn from the account (Withdrawn amount should be given as input). For withdrawal the condition is- the AvlBalance must be greater than withdrawn amount).

   2 is chosen- Amount can be deposited to the account (the deposited amount should be given as input). The deposited amount should be reflected in AvlBalance of the account.

   3 is chosen- Current available balance (AvlBalance) of the AccNumber should be Displayed with other details

   4 is chosen- Exit from the application

   Sample Input:

   SB 100155 VenkatKrishna 4500.00 Saving

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

struct BankMgmt {
    int AccNumber;
    char CustName[50];
    float AvlBalance;
    char AccType[20];
};

void withdraw(struct BankMgmt *account) {
    float amount;
    printf("Enter amount to withdraw: ");
    scanf("%f", &amount);

    if (account->AvlBalance >= amount) {
        account->AvlBalance -= amount;
        printf("Withdrawal successful. New balance: %.2f\n", account->AvlBalance);
    } else {
        printf("Insufficient balance.\n");
    }
}

void deposit(struct BankMgmt *account) {
    float amount;
    printf("Enter amount to deposit: ");
    scanf("%f", &amount);

    account->AvlBalance += amount;
    printf("Deposit successful. New balance: %.2f\n", account->AvlBalance);
}

void displayBalance(struct BankMgmt *account) {
    printf("Account Number: %d\n", account->AccNumber);
    printf("Customer Name: %s\n", account->CustName);
    printf("Available Balance: %.2f\n", account->AvlBalance);
    printf("Account Type: %s\n", account->AccType);
}

int main() {
    struct BankMgmt account;
    int choice;

    account.AccNumber = 100155;
    strcpy(account.CustName, "VenkatKrishna");
    account.AvlBalance = 4500.00;
    strcpy(account.AccType, "Saving");

    do {
        printf("\nMenu:\n");
        printf("1. Withdrawal\n");
        printf("2. Deposit\n");
        printf("3. Display Balance\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                withdraw(&account);
                break;
            case 2:
                deposit(&account);
                break;
            case 3:
                displayBalance(&account);
                break;
            case 4:
                printf("Exiting the application.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```