1.

```
/*
typedef is a keyword: this is used to provide an alias
or a new name to an already existing data type

C syntac for typedef

typedef existing_name_of_the_data_type alias_name;

example
typedef double dbl;


USE CASES :
1.Can be used for already existing data types
2.Can be used for user defined datatypes such as structures which improves the
readability of the code.
3.Can be used with pointers
4.Is is compactable with arrays as well


*/
#include <stdio.h>

typedef int my_int;

int main()
{
    //alias name my_int has been used for declaring the variable
    my_int a = 28;
    printf("a = %d\n", a);
    return 0;
}
```

2.

```
//Implementation of struct using typedef
#include <stdio.h>

typedef struct date{
    int day;
    int month;
    int year;
}dt;

int main()
{
    //alias name dt has been used for declaring the variable
```

```
    dt var1 = {26,11,2024};


    printf("sizeof var1  = %d\n", sizeof(var1));

    printf("\nToday's Date = %d-%d-%d",var1.day,var1.month,var1.year);
    return 0;
}
```

3.

```
//using typedef with pointers

#include <stdio.h>

typedef int* ptr;

int main()
{
    //alias name ptr has been used for declaring the variable
    int a = 28;
    ptr ptr1 = &a;
    printf("1...a = %d\n", *ptr1);

    *ptr1 = 30;

    printf("2...a = %d\n", *ptr1);
    return 0;
}
```

4.

```
//using typedef with array

#include <stdio.h>

typedef int arr[4];

int main()
{
    //alias name arr has been used for declaring the variable
    arr a = {1,2,3,4};
    for(int i=0;i<4;i++)
    {
        printf("%d ", a[i]);
    }

    return 0;
}
```

5.

```c
/*
Problem Statement:
Write a program that defines a custom data type Complex using typedef to
represent a complex number with real and imaginary parts. Implement functions
to:
•   Add two complex numbers.
•   Multiply two complex numbers.
•   Display a complex number in the format "a + bi".
Input Example
Enter first complex number (real and imaginary): 3 4
Enter second complex number (real and imaginary): 1 2
Output Example
Sum: 4 + 6i
Product: -5 + 10


*/
#include <stdio.h>

// Define a custom data type for Complex numbers using typedef
typedef struct {
    int real;
    int imag;
} Complex;

// Function to add two complex numbers
Complex add(Complex c1, Complex c2) {
    Complex result;
    result.real = c1.real + c2.real;
    result.imag = c1.imag + c2.imag;
    return result;
}

// Function to multiply two complex numbers
Complex multiply(Complex c1, Complex c2) {
    Complex result;
    result.real = c1.real * c2.real - c1.imag * c2.imag;
    result.imag = c1.real * c2.imag + c1.imag * c2.real;
    return result;
}

// Function to display a complex number
void display(Complex c) {
    printf("%d + %di\n", c.real, c.imag);
}

int main() {
    Complex c1, c2, sum, product;
```

```
    // Input first complex number
    printf("Enter first complex number (real and imaginary): ");
    scanf("%d %d", &c1.real, &c1.imag);

    // Input second complex number
    printf("Enter second complex number (real and imaginary): ");
    scanf("%d %d", &c2.real, &c2.imag);

    // Perform addition and multiplication
    sum = add(c1, c2);
    product = multiply(c1, c2);

    // Display results
    printf("Sum: ");
    display(sum);
    printf("Product: ");
    display(product);

    return 0;
}
```

6.

```
/*
Typedef for Structures
Problem Statement:
Define a custom data type Rectangle using typedef to represent a rectangle
with width and height as float values. Write functions to:
•   Compute the area of a rectangle.
•   Compute the perimeter of a rectangle.
Input Example:
Enter width and height of the rectangle: 5 10
Output Example:
Area: 50.00
Perimeter: 30.00


*/
#include <stdio.h>

// Define a custom data type for Rectangle using typedef
typedef struct {
```

```c
    float width;
    float height;
} Rectangle;

// Function to compute the area of a rectangle
float computeArea(Rectangle r) {
    return r.width * r.height;
}

// Function to compute the perimeter of a rectangle
float computePerimeter(Rectangle r) {
    return 2 * (r.width + r.height);
}

int main() {
    Rectangle rect;
    float area, perimeter;

    // Input width and height of the rectangle
    printf("Enter width and height of the rectangle: ");
    scanf("%f %f", &rect.width, &rect.height);

    // Compute area and perimeter
    area = computeArea(rect);
    perimeter = computePerimeter(rect);

    // Display results
    printf("Area: %.2f\n", area);
    printf("Perimeter: %.2f\n", perimeter);

    return 0;
}
```

7.

```c
/*Funtion Pointers  */


#include <stdio.h>

void display(int);

int main(){
    //Declaration a pointer to the function display()
    void (*func_ptr)(int);
    //Initializing the pointer with the address of function display()
    func_ptr = &display;
```

```
    //Calling the function as well passing the parameter using function
pointers
    (*func_ptr)(20);
    return 0;
}

void display(int a){
    printf("a = %d",a);
}
```

8.

```
/*
7. Problem Statement: Vehicle Management System
Write a C program to manage information about various vehicles. The program
should demonstrate the following:
1.  Structures: Use structures to store common attributes of a vehicle, such
as vehicle type, manufacturer name, and model year.
2.  Unions: Use a union to represent type-specific attributes, such as:
o   Car: Number of doors and seating capacity.
o   Bike: Engine capacity and type (e.g., sports, cruiser).
o   Truck: Load capacity and number of axles.
3.  Typedefs: Define meaningful aliases for complex data types using typedef
(e.g., for the structure and union types).
4.  Bitfields: Use bitfields to store flags for vehicle features like airbags,
ABS, and sunroof.
5.  Function Pointers: Use a function pointer to dynamically select a function
to display specific information about a vehicle based on its type.
Requirements
1.  Create a structure Vehicle that includes:
o   A char array for the manufacturer name.
o   An integer for the model year.
o   A union VehicleDetails for type-specific attributes.
o   A bitfield to store vehicle features (e.g., airbags, ABS, sunroof).
o   A function pointer to display type-specific details.
2.  Write functions to:
o   Input vehicle data, including type-specific details and features.
o   Display all the details of a vehicle, including the type-specific
attributes.
o   Set the function pointer based on the vehicle type.
3.  Provide a menu-driven interface to:
o   Add a vehicle.
o   Display vehicle details.
o   Exit the program.


Example Input/Output
Input:
```

```
1. Add Vehicle
2. Display Vehicle Details
3. Exit
Enter your choice: 1

Enter vehicle type (1: Car, 2: Bike, 3: Truck): 1
Enter manufacturer name: Toyota
Enter model year: 2021
Enter number of doors: 4
Enter seating capacity: 5
Enter features (Airbags[1/0], ABS[1/0], Sunroof[1/0]): 1 1 0

1. Add Vehicle
2. Display Vehicle Details
3. Exit
Enter your choice: 2

Output:
Manufacturer: Toyota
Model Year: 2021
Type: Car
Number of Doors: 4
Seating Capacity: 5
Features: Airbags: Yes, ABS: Yes, Sunroof: No




*/



#include <stdio.h>
#include <string.h>

// Typedef for Vehicle Features using Bitfields
typedef struct {
    unsigned int airbags : 1;
    unsigned int abs : 1;
    unsigned int sunroof : 1;
} Features;

// Union for Type-Specific Attributes
typedef union {
    struct {
        int doors;
        int seating_capacity;
    } car;
    struct {
```

```
        int engine_capacity;
        char type[20]; // e.g., "sports", "cruiser"
    } bike;
    struct {
        int load_capacity;
        int axles;
    } truck;
} VehicleDetails;

// Structure for Vehicle
typedef struct {
    char manufacturer[50];
    int model_year;
    int type; // 1: Car, 2: Bike, 3: Truck
    VehicleDetails details;
    Features features;
    void (*displayDetails)(void *); // Function pointer to display details
} Vehicle;

// Function to display car details
void displayCarDetails(void *details) {
    VehicleDetails *carDetails = (VehicleDetails *)details;
    printf("Type: Car\n");
    printf("Number of Doors: %d\n", carDetails->car.doors);
    printf("Seating Capacity: %d\n", carDetails->car.seating_capacity);
}

// Function to display bike details
void displayBikeDetails(void *details) {
    VehicleDetails *bikeDetails = (VehicleDetails *)details;
    printf("Type: Bike\n");
    printf("Engine Capacity: %d cc\n", bikeDetails->bike.engine_capacity);
    printf("Type: %s\n", bikeDetails->bike.type);
}

// Function to display truck details
void displayTruckDetails(void *details) {
    VehicleDetails *truckDetails = (VehicleDetails *)details;
    printf("Type: Truck\n");
    printf("Load Capacity: %d kg\n", truckDetails->truck.load_capacity);
    printf("Number of Axles: %d\n", truckDetails->truck.axles);
}

// Function to input vehicle data
void addVehicle(Vehicle *v) {
    printf("Enter manufacturer name: ");
    scanf(" %[^\n]", v->manufacturer);
    printf("Enter model year: ");
```

```
    scanf("%d", &v->model_year);

    printf("Enter vehicle type (1: Car, 2: Bike, 3: Truck): ");
    scanf("%d", &v->type);

    if (v->type == 1) {
        printf("Enter number of doors: ");
        scanf("%d", &v->details.car.doors);
        printf("Enter seating capacity: ");
        scanf("%d", &v->details.car.seating_capacity);
        v->displayDetails = displayCarDetails;
    } else if (v->type == 2) {
        printf("Enter engine capacity (cc): ");
        scanf("%d", &v->details.bike.engine_capacity);
        printf("Enter type (e.g., sports, cruiser): ");
        scanf(" %[^\n]", v->details.bike.type);
        v->displayDetails = displayBikeDetails;
    } else if (v->type == 3) {
        printf("Enter load capacity (kg): ");
        scanf("%d", &v->details.truck.load_capacity);
        printf("Enter number of axles: ");
        scanf("%d", &v->details.truck.axles);
        v->displayDetails = displayTruckDetails;
    } else {
        printf("Invalid type!\n");
        return;
    }

    printf("Enter features (Airbags[1/0], ABS[1/0], Sunroof[1/0]): ");
    scanf("%u %u %u", &v->features.airbags, &v->features.abs, &v-
>features.sunroof);
}

// Function to display vehicle details
void displayVehicle(Vehicle *v) {
    printf("Manufacturer: %s\n", v->manufacturer);
    printf("Model Year: %d\n", v->model_year);
    if (v->displayDetails != NULL) {
        v->displayDetails(&v->details);
    }
    printf("Features: Airbags: %s, ABS: %s, Sunroof: %s\n",
           v->features.airbags ? "Yes" : "No",
           v->features.abs ? "Yes" : "No",
           v->features.sunroof ? "Yes" : "No");
}

// Main function with a menu-driven interface
int main() {
```

```
    Vehicle vehicle;
    int choice;

    while (1) {
        printf("\n1. Add Vehicle\n2. Display Vehicle Details\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addVehicle(&vehicle);
                break;
            case 2:
                displayVehicle(&vehicle);
                break;
            case 3:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

9.

```
/*Funtion Pointers  */

/*
#include <stdio.h>

void display(int);

int main(){
    //Declaration a pointer to the function display()
    void (*func_ptr)(int);
    //Initializing the pointer with the address of function display()
    func_ptr = &display;
    //Calling the function as well passing the parameter using function
pointers
    (*func_ptr)(20);
    return 0;
}

void display(int a){
    printf("a = %d",a);
}
```

```c
*/

//Array of function pointers

#include <stdio.h>

void add(int , int);
void sub(int , int);
void mul(int , int);

int main(){
    void (*fun_ptr_arr[])(int, int) = {add, sub, mul};
    int a = 10, b = 20;
    (*fun_ptr_arr[0])(a,b);
    (*fun_ptr_arr[1])(a,b);
    (*fun_ptr_arr[2])(a,b);
    return 0;
}

void add(int a, int b){
    int sum = a + b;
    printf("sum = %d \n",sum);
}
void sub(int a, int b){
    int sub = a - b;
    printf("sub = %d \n",sub);
}
void mul(int a, int b){
    int mul = a * b;
    printf("mul = %d \n",mul);
}
```

10.

```c
/*
Recursion: A Function calling itself

Basic synatx for Recursive Function

return funtion_name(args..){
    //base(exit) condition
    //recursion call funtion_name(args);
}

WAP to caluclate the sum of first N natural numbers using recurion
```

```c
*/

#include <stdio.h>

int sumNatural(int);

int main(){
    int n;
    printf("Enter The limit till which the summation of natural number should
happen: ");
    scanf("%d",&n);
    printf("\n");
    int sum = sumNatural(n);
    printf("sum = %d",sum);
    return 0;
}

int sumNatural(int n){
    int res = 0;
    //base condition
    if(n == 0){
        return 0;
    }
    //recursive call
    res = n + sumNatural(n-1);
    return res;
}
```

11.

```c
//WAP to find out the factorial of a number using recursion.

#include <stdio.h>
int factNum(int);
int main()
{
    int n;
    printf("\nEnter the number : ");
    scanf("%d",&n);
    int result = factNum(n);
    printf("\nThe factorial of %d is %d",n,result);
    return 0;
}
int factNum(int num)
{

    if(num>=1)
    {
```

```
        return num * factNum(num-1);
    }
    else
        return 1;
}
```

12.

```
//2. WAP to find the sum of digits of a number using recursion.
#include <stdio.h>
int sumDigits(int);
int main()
{
    int n;
    printf("\nEnter the number : ");
    scanf("%d",&n);
    int result = sumDigits(n);
    printf("\nThe sum of digits of %d is %d",n,result);
    return 0;
}
int sumDigits(int num)
{

    if(num==0)
        return 0;
    else
    {

        return (num%10)+sumDigits(num/10);
    }

}
```

13.

```
//3. With Recursion Findout the maximum number in a given array

#include <stdio.h>
int maxElmnt(int arr[],int size);
int main()
{
    int arr[] = {1,2,3,4};
    int size = sizeof(arr)/sizeof(arr[0]);

    int result = maxElmnt(arr,size);
    printf("\nThe the maximum number in a given array is %d",result);
    return 0;
}
```

```c
int maxElmnt(int arr[],int size)
{

    if(size==1)
        return arr[0];
    int max = maxElmnt(arr,size-1);
    return (arr[size-1]>max)?arr[size-1]:max;

}
```

14.

```c
//4. With recurion calculate the power of a given number


#include <stdio.h>

int power(int base, int exponent);

int main() {
    int base, exponent;

    printf("Enter the base: ");
    scanf("%d", &base);
    printf("Enter the exponent: ");
    scanf("%d", &exponent);

    int result = power(base, exponent);
    printf("\nThe result of %d^%d is %d\n", base, exponent, result);

    return 0;
}

// Recursive function to calculate the power of a number
int power(int base, int exponent) {
    // Base case: any number raised to the power 0 is 1
    if (exponent == 0) {
        return 1;
    }

    // Recursive case: base * base^(exponent - 1)
    return base * power(base, exponent - 1);
}
```

15.

```c
//5.
```

```c
#include <stdio.h>

int stringLength(char *str);

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str); // Reads a string

    int length = stringLength(str);
    printf("The length of the string \"%s\" is %d\n", str, length);

    return 0;
}

// Recursive function to calculate the length of a string
int stringLength(char *str) {
    // Base case: If the string is empty (null character), return 0
    if (*str == '\0') {
        return 0;
    }

    // Recursive case: Move to the next character and add 1 to the result
    return 1 + stringLength(str + 1);
}
```

16.

```c
//6. With recursion revrsal of a string
#include <stdio.h>

void reverseString(char *str, int start, int end);

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    int length = 0;
    while (str[length] != '\0') {
        length++;
    }

    reverseString(str, 0, length - 1);
```

```
    printf("Reversed string: %s\n", str);

    return 0;
}

// Recursive function to reverse a string
void reverseString(char *str, int start, int end) {
    // Base case: If start is greater or equal to end, do nothing
    if (start >= end) {
        return;
    }

    // Swap characters at start and end
    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;

    // Recursive call for the rest of the string
    reverseString(str, start + 1, end - 1);
}
```

17.

```
/*Funtion Pointers */
#include <stdio.h>
void display(int);
int main(){
//Declaration a pointer to the function display()
void (*func_ptr)(int);
//Initializing the pointer with the address of function display()
func_ptr = &display;
//Calling the function as well passing the parameter using function pointers
(*func_ptr)(20);
return 0;
}
void display(int a){
printf("a = %d",a);
}
```

18.

```
/*Array of Function pointers*/
#include <stdio.h>
void add(int, int);
void sub(int, int);
void mul(int, int);
int main(){
```

```
 void(*fun_ptr_arr[])(int,int) = {add,sub,mul};

 int a = 10,b = 20;
 (*fun_ptr_arr[0])(a,b);
 (*fun_ptr_arr[1])(a,b);
 (*fun_ptr_arr[2])(a,b);

 return 0;

}
void add(int a, int b){
 int sum = a+b;
 printf("sum = %d\n",sum);
}
void sub(int a, int b){
 int sub = a-b;
 printf("Sub is %d\n",sub);
}
void mul(int a, int b){
 int mul = a*b;
 printf("Mul is %d\n",mul);
}
```

19.

```
/*Simple Calculator Using Function Pointers
Problem Statement:
Write a C program to implement a simple calculator. Use function pointers to
dynamically call functions for addition, subtraction, multiplication, and
division based
on user input.
Input Example:
Enter two numbers: 10 5
Choose operation (+, -, *, /): *
Output Example:
Result: 50*/
#include <stdio.h>
void add(int, int);
void sub(int, int);
void mul(int, int);
void divi(int, int);
int main(){

 void(*fun_ptr_arr[])(int,int) = {add,sub,mul,divi};

 int a,b;
 printf("Enter two numbers:");
```

```
 scanf("%d %d",&a,&b);

 char op;
 printf("Choose operation(+, -, *, /):");
 getchar();
 scanf("%c",&op);
 switch(op){
 case '+':
 (*fun_ptr_arr[0])(a,b);
 break;
 case '-':
 (*fun_ptr_arr[1])(a,b);
 break;
 case '*':
 (*fun_ptr_arr[2])(a,b);
 break;
 case '/':
 (*fun_ptr_arr[3])(a,b);
 break;
 default:
 printf("Invalid option!\n");
 }




 return 0;

}
void add(int a, int b){
 int sum = a+b;
 printf("sum = %d\n",sum);
}
void sub(int a, int b){
 int sub = a-b;
 printf("Sub is %d\n",sub);
}
void mul(int a, int b){
 int mul = a*b;
 printf("Mul is %d\n",mul);
}
void divi(int a, int b){
 int divi = a/b;
 printf("Div is %d\n",divi);
}
```

20.

```c
/*Array Operations Using Function Pointers
Problem Statement:
Write a C program that applies different operations to an array of integers
using
function pointers. Implement operations like finding the maximum, minimum, and
sum of elements.
Input Example:
Enter size of array: 4
Enter elements: 10 20 30 40
Choose operation (1 for Max, 2 for Min, 3 for Sum): 3
Output Example:
Result: 100*/
#include <stdio.h>
int max(int [],int);
int min(int [],int);
int sum(int [],int);
int main(){

 int (*fun_ptr_arr[])(int [],int) = {max,min,sum};

 int size;
 printf("Enter the size of array:");
 scanf("%d",&size);
 int arr[size];
 printf("Enter the array elements: ");
 for(int i=0;i<size;i++)
 {
 scanf("%d",&arr[i]);
 }
 int op;
 printf("Choose operation (1 for Max, 2 for Min, 3 for Sum):");
 scanf("%d",&op);

 switch(op){
 case 1:
 int max = (*fun_ptr_arr[0])(arr,size);
 printf("MAximum is %d\n",max);
 break;
 case 2:
 int min = (*fun_ptr_arr[1])(arr,size);
 printf("Minimum is %d\n",min);
 break;
 case 3:
 int sum = (*fun_ptr_arr[2])(arr,size);
 printf("Sum is %d\n",sum);
 break;

 default:
```

```
 printf("Invalid option!\n");
 }

}
int max(int arr[],int s){
 int max=arr[0];
 for(int i=1;i<s;i++)
 {
 if(max < arr[i])
 {
 max = arr[i];
 }
 }
 return max;
}
int min(int arr[],int s){
 int min=arr[0];
 for(int i=1;i<s;i++)
 {
 if(min > arr[i])
 {
 min = arr[i];
 }
 }
 return min;
}
int sum(int arr[],int s){
 int sum_1=0;
 for(int i=0;i<s;i++)
 {
 sum_1 += arr[i];
 }
 return sum_1;
}
```

21.

```
/*Event System Using Function Pointers
Problem Statement:
Write a C program to simulate a simple event system. Define three events:
onStart,
onProcess, and onEnd. Use function pointers to call appropriate event handlers
dynamically based on user selection.*/
#include <stdio.h>
void onStart();
void onProcess();
void onEnd();
int main() {
```

```
 void (*eventHandlers[])() = {onStart, onProcess, onEnd};
 int event;
 printf("Choose event (1 for onStart, 2 for onProcess, 3 for onEnd): ");
 scanf("%d", &event);

 if (event >= 1 && event <= 3) {
 printf("Event: ");
 switch (event) {
 case 1:
 printf("onStart\n");
 eventHandlers[event - 1]();
 break;
 case 2:
 printf("onProcess\n");
 eventHandlers[event - 1]();
 break;
 case 3:
 printf("onEnd\n");
 eventHandlers[event - 1]();
 break;
 }

 } else {
 printf("Invalid event selection!\n");
 }
 return 0;
}
void onStart() {
 printf("Starting the process...\n");
}
void onProcess() {
 printf("Processing the data...\n");
}
void onEnd() {
 printf("Ending the process...\n");
}
```

22.

```
Matrix Operations with Function Pointers
Problem Statement:
Write a C program to perform matrix operations using function pointers.
Implement
functions to add, subtract, and multiply matrices. Pass the function pointer
to a
wrapper function to perform the desired operation.
#include <stdio.h>
```

```c
#include <stdlib.h>
void add(int **mat1, int **mat2, int r, int c);
void sub(int **mat1, int **mat2, int r, int c);
void mul(int **mat1, int **mat2, int r, int c);
int main() {

 void (*fun_ptr_arr[3])(int **mat1, int **mat2, int, int) = {add, sub, mul};

 int r, c, op;
 printf("Enter matrix size (rows and columns): ");
 scanf("%d %d", &r, &c);


 int **mat1 = (int **)malloc(r * sizeof(int *));
 int **mat2 = (int **)malloc(r * sizeof(int *));

 for (int i = 0; i < r; i++) {
 mat1[i] = (int *)malloc(c * sizeof(int));
 mat2[i] = (int *)malloc(c * sizeof(int));
 }


 printf("Enter first matrix:\n");
 for (int i = 0; i < r; i++)
 {
 for (int j = 0; j < c; j++)
 {
 scanf("%d", &mat1[i][j]);
 }
 }


 printf("Enter second matrix:\n");
 for (int i = 0; i < r; i++)
 {
 for (int j = 0; j < c; j++)
 {
 scanf("%d", &mat2[i][j]);
 }
 }


 printf("Choose operation (1 for Add, 2 for Subtract, 3 for Multiply): ");
 scanf("%d", &op);

 switch (op) {
 case 1:
 (*fun_ptr_arr[0])(mat1, mat2, r, c);
 break;
```

```
 case 2:
 (*fun_ptr_arr[1])(mat1, mat2, r, c);
 break;
 case 3:
 (*fun_ptr_arr[2])(mat1, mat2, r, c);
 break;
 default:
 printf("Invalid option!!\n");
 }


 for (int i = 0; i < r; i++)
 {
 free(mat1[i]);
 free(mat2[i]);
 }
 free(mat1);
 free(mat2);

 return 0;
}
void add(int **mat1, int **mat2, int r, int c)
{
 printf("Result:\n");
 for (int i = 0; i < r; i++)
 {
 for (int j = 0; j < c; j++)
 {
 printf("%d ", mat1[i][j] + mat2[i][j]);
 }
 printf("\n");
 }
}
void sub(int **mat1, int **mat2, int r, int c)
{
 printf("Result:\n");
 for (int i = 0; i < r; i++)
 {
 for (int j = 0; j < c; j++)
 {
 printf("%d ", mat1[i][j] - mat2[i][j]);
 }
 printf("\n");
 }
}
void mul(int **mat1, int **mat2, int r, int c)
{
 int **result = (int **)malloc(r * sizeof(int *));
 for (int i = 0; i < r; i++)
```

```
{
result[i] = (int *)malloc(c * sizeof(int));
}


printf("Result:\n");
for (int i = 0; i < r; i++)
{
for (int j = 0; j < c; j++)
{
result[i][j] = 0;
for (int k = 0; k < c; k++)
{
result[i][j] += mat1[i][k] * mat2[k][j];
}
printf("%d ", result[i][j]);
}
printf("\n");
}

for (int i = 0; i < r; i++) {
free(result[i]);
}
free(result);
}
```