

1. Write a C program to determine if the given number is odd or even using Bitwise operators.

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num & 1) {
        printf("%d is an odd number\n", num);
    } else {
        printf("%d is an even number\n", num);
    }
    return 0;
}
```

OUTPUTA screenshot of a terminal window with a dark background. The title bar says 'Output' and there is a 'Clear' button. The terminal shows the prompt '/tmp/pTikG3mYnJ.o', followed by the user input 'Enter a number: 103' and the program output '103 is an odd number'.**2. Write a C program to count the number of bits set in a number.**

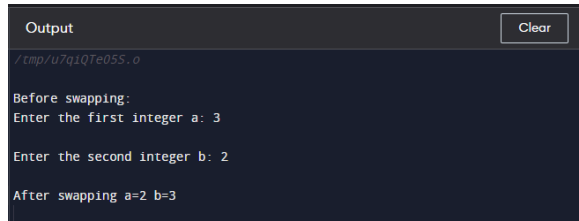
```
#include <stdio.h>
int main() {
    int num, count=0;
    printf("Enter a number: ");
    scanf("%d", &num);
    while (num) {
        count += num & 1;
        num >>= 1;
    }
    printf("Count of Set bits: %d\n", count);

    return 0;
}
```

OUTPUTA screenshot of a terminal window with a dark background. The title bar says 'Output' and there is a 'Clear' button. The terminal shows the prompt '/tmp/u7q1QTe05S.o', followed by the user input 'Enter a number: 144' and the program output 'Count of Set bits: 2'.

3. Write a C program to swap two numbers. Use a function pointer to do this operation.

```
#include <stdio.h>
int swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
    return *a,*b;
}
int main() {
    int a, b;
    printf("\nBefore swapping: ");
    printf("\nEnter the first integer a: ");
    scanf("%d", &a);
    printf("\nEnter the second integer b: ");
    scanf("%d", &b);
    swap(&a, &b);
    printf("\nAfter swapping a=%d b=%d", a, b);
    return 0;
}
```

OUTPUT


```
Output
/tmp/u7q1QTe055.o
Before swapping:
Enter the first integer a: 3
Enter the second integer b: 2
After swapping a=2 b=3
```

4. Write an equivalent pointer expression for fetching the value of array element $a[i][j][k][2]$ **ANSWER**

$*(*(*(a + i) + j) + k) + 2)$

- a is a pointer and i , j , k , l are offset values.
- $\text{int ptr} = a + i$ where a is Base address and i is Offset, now it gives the address of $a[i]$ which is in integer form.
- Hence we will it into address as $*(a+i)$ and then add it with the 2-nd offset i.e. j -> $*(a+i)+j$.
- Similarly The Process Continues as Follows:
- Address = $*(a+i)+j$ Offset = k
- new Address = $*(*(a+i)+j)+k$
- Address = $*(*(a+i)+j)+k$ Offset = l

- new Address = $*(*(a+i)+j)+k)+l$

And Then Finally The Address Of The Element $a[i][j][k][l] = *(*(*(a+i)+j)+k)+l$.

5. Write a C program to Multiply two matrix (n*n) using pointers.

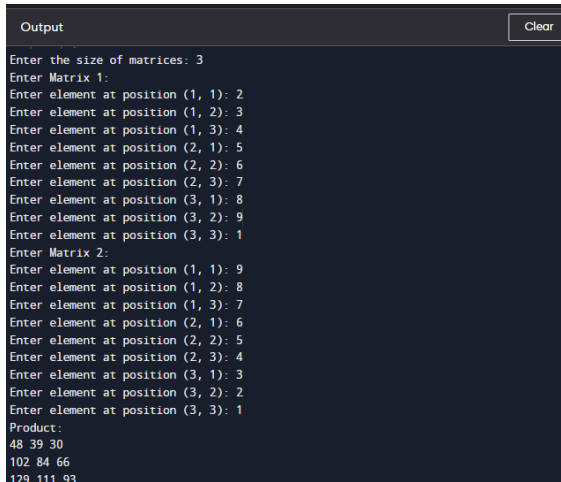
```
#include<stdio.h>
void multiplyMatrices(int n, int (*mat1)[n], int (*mat2)[n], int (*result)[n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = 0;
            for (int k = 0; k < n; k++) {
                result[i][j] += (*(mat1 + i) + k) * (*(mat2 + k) + j);
            }
        }
    }
}
void displayMatrix(int n, int (*matrix)[n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", (*(matrix + i) + j));
        }
        printf("\n");
    }
}
int main() {
    int n;
    printf("Enter the size of matrices: ");
    scanf("%d", &n);
    int matrix1[n][n], matrix2[n][n], result[n][n];
    printf("Enter Matrix 1:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("Enter element at position (%d, %d): ", i + 1, j + 1);
            scanf("%d", &matrix1[i][j]);
        }
    }
    printf("Enter Matrix 2:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("Enter element at position (%d, %d): ", i + 1, j + 1);
            scanf("%d", &matrix2[i][j]);
        }
    }
    multiplyMatrices(n, matrix1, matrix2, result);
```

```

printf("Product:\n");
displayMatrix(n, result);
return 0;
}

```

OUTPUT



```

Output
Enter the size of matrices: 3
Enter Matrix 1:
Enter element at position (1, 1): 2
Enter element at position (1, 2): 3
Enter element at position (1, 3): 4
Enter element at position (2, 1): 5
Enter element at position (2, 2): 6
Enter element at position (2, 3): 7
Enter element at position (3, 1): 8
Enter element at position (3, 2): 9
Enter element at position (3, 3): 1
Enter Matrix 2:
Enter element at position (1, 1): 9
Enter element at position (1, 2): 8
Enter element at position (1, 3): 7
Enter element at position (2, 1): 6
Enter element at position (2, 2): 5
Enter element at position (2, 3): 4
Enter element at position (3, 1): 3
Enter element at position (3, 2): 2
Enter element at position (3, 3): 1
Product:
48 39 30
102 84 66
129 111 93

```

6. Find the output of the following // Consider the compiler is 32-bit machine

```

#include <stdio.h>
typedef struct
{
int A;
char B;
char C;
} InfoData;
int main(int argc, char *argv[])
{ //Calculate size of structure
printf("\n Size of Structure = %d\n\n",sizeof(InfoData));
return 0;
}

```

ANSWER

In this example, the output indicates that the size of the InfoData structure is 8 bytes. The breakdown of the size is as follows:

- int A takes 4 bytes (on a typical 32-bit machine).
- char B takes 1 byte.
- char C takes 1 byte.
- The compiler may add 2 bytes of padding to align the structure.

So, the total size is $4 + 1 + 1 + 2 = 8$ bytes. The size of a structure depends on the size and alignment requirements of its individual members.

OUTPUT


```

Output
/tmp/u7q1QTe055.o
Size of Structure = 8
  
```

7. Find the output of the following // Consider the compiler is 32-bit machine

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

```
typedef struct {
    char A;
    double B;
    char C;
} InfoData;
```

```
int main(int argc, char *argv[]) {
    // Calculate size of structure
    printf("\n Size of Structure = %d\n\n", sizeof(InfoData));
    return 0;
}
```

ANSWER

In InfoData structure (on a typical 32-bit machine).

char A takes 1 bytes

Padding after char A to ensure alignment for double B takes 7 bytes

double B takes 8 byte.

char C takes 1 byte.

Additional padding to ensure overall structure alignment takes 7 bytes

Summing these up is $1 + 7 + 8 + 1 + 7 = 24$ bytes.

OUTPUT


```

Output
/tmp/yQ10LEk6CJ.o
Size of Structure = 24
  
```

8. Find the output of the following // Consider the compiler is 32-bit machine

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

```
#include <stdint.h>
int main(){
    unsigned int var = 0x12345678;
    unsigned int rev = 0;
    for (int i = 0; i < 8; i++){
        rev = (rev << 4) | ((var >> (4*i)) & 0xF);
    }
    printf("%x", rev);
    return 0;
}
```

OUTPUT

In each iteration, the value of rev is shifted left by 4 bits, and a new set of 4 bits is extracted from var and combined with the existing rev. This process is repeated for 8 iterations, resulting in the reversed value 0x87654321.

