**Problem 1:**

**Write a program to find out the G.C.D and L.C.M of two numbers.**

**Code:**

#include<stdio.h>

int main()

{

    int a,b,t,x,y,z,LCM;

    printf("enter two numbers(a b): ");

    scanf("*%d* *%d*",&a,&b);

    x = a,y = b;

    while(a!=0)

    {

        t = a;

        a = b%a;

        b = t;

    }

    printf("value of G.C.D is *%d*\n",b);

    z = x\*y;

    LCM = z/b;

    printf("value of L.C.M is *%d*\n",LCM);

    return 0;

}

**Output:**

**enter two numbers(a b): 15 17  
value of G.C.D is 1  
value of L.C.M is 255**

**enter two numbers(a b): 12 14  
value of G.C.D is 2  
value of L.C.M is 84**

**Conclusion:**

**The provided C code calculates the Greatest Common Divisor (GCD) and the Least Common Multiple (LCM) of two input numbers using the Euclidean algorithm. The user is prompted to input two integers, and the program proceeds to find their GCD through a while loop. Once the GCD is determined, the LCM is calculated using the relationship between GCD and LCM. The final results, GCD and LCM, are then displayed. The code appears to be functional for computing GCD and LCM and can serve as a basic utility for these mathematical operations.**

**Problem 2:**

**Write a program to reverse a number.**

**Code:**

#include<stdio.h>

int main()

{

    int a,sum=0;

    int rem;

    printf("enter the number to reverse: ");

    scanf("*%d*",&a);

    while(a != 0)

    {

        rem = a%10;

        sum = (10\*sum)+rem;

        a = (int)a/10;

    }

    printf("reversed number is: *%d*",sum);

    return 0;

}

**Output:**

**enter the number to reverse: 54275  
reversed number is: 57245**

**enter the number to reverse: 9420579  
reversed number is: 9750249**

**Conclusion:**

**In conclusion, the provided C code is designed to reverse a given integer. The program prompts the user to input a number, and then it uses a while loop to iteratively extract the last digit of the number, accumulate it in a new reversed number, and remove the last digit from the original number. The process continues until the original number becomes zero. Finally, the reversed number is displayed as the output. The code appears to be correctly implemented for reversing an integer, demonstrating a common technique using a while loop and simple arithmetic operations.**

**Problem 3:**

**Write a program to find out the the area of a traingle, square and rectangle.**

**Code:**

#include<stdio.h>

#include<math.h>

#define AND &&

#define OR ||

float area\_tri(float *a*,float *b*,float *c*);

int main()

{

    int choice;

    printf("enter which area do you want to get: \n");

    printf("1 for rectangle,2 for square,3 for triangle: ");

    scanf("*%d*",&choice);

    if (choice == 1)

    {

*//rectangle*

        float length,breadth;

        printf("enter the value of length: ");

        scanf("*%f*",&length);

        printf("enter the value of breadth: ");

        scanf("*%f*",&breadth);

        printf("the area of the rectangle is: *%f*",length\*breadth);

    }

    else if(choice == 2)

    {

        float side;

        printf("enter the value of one side: ");

        scanf("*%f*",&side);

        printf("the area of the square is: *%f*",side\*side);

    }

    else if(choice == 3)

    {

        float a,b,c;

        printf("enter the value of three sides of the triangle:\n");

        printf("a: ");

        scanf("*%f*",&a);

        printf("b: ");

        scanf("*%f*",&b);

        printf("c: ");

        scanf("*%f*",&c);

        if (a+b>c AND a+c>b AND c+b>a)

            printf("the area of the triangle is *%f*",area\_tri(a,b,c));

        else

            printf("triangle can not be created");

    }

    else

        printf("invalid input");

}

float area\_tri(float *a*,float *b*,float *c*)

{

    float s,area;

    s = (*a*+*b*+*c*)/2;

    area = sqrt(s\*(s-*a*)\*(s-*b*)\*(s-*c*));

    return area;

}

**Output:**

**Case 1:**

**enter which area do you want to get:   
1 for rectangle,2 for square,3 for triangle: 1  
enter the value of length: 5.45  
enter the value of breadth: 3.25  
the area of the rectangle is: 17.712499**

**case 2:**

**enter which area do you want to get:   
1 for rectangle,2 for square,3 for triangle: 2  
enter the value of one side: 3.54  
the area of the square is: 12.531600**

**case 3:**

**enter which area do you want to get:  
1 for rectangle,2 for square,3 for triangle: 3  
enter the value of three sides of the triangle:  
a: 6  
b: 4  
c: 15  
triangle can not be created**

**case 4:**

**enter which area do you want to get:   
1 for rectangle,2 for square,3 for triangle: 3  
enter the value of three sides of the triangle:  
a: 6  
b: 4  
c: 5  
the area of the triangle is 9.921567**

**Conclusion:**

**In conclusion, the provided C code is a simple program designed to calculate the area of a geometric shape based on user input. The user is prompted to choose the type of area they want to calculate—either for a rectangle, a square, or a triangle. The program then takes the necessary input values, performs the corresponding calculations, and prints the result.**

**The code incorporates conditional statements to handle different cases for rectangle, square, and triangle. For the triangle, there is an additional check to ensure the input values can form a valid triangle using the triangle inequality theorem. If the conditions are met, the program calls a separate function to calculate the area of the triangle using Heron's formula.**

**Overall, the code provides a clear and organized structure for computing areas of different shapes and includes appropriate input validation for the triangle case.**

**Problem 4:**

**Write a program to generate the following patterns for N number of rows:**

**i)**

**\***

**\* \***

**\* \* \***

**\* \* \* \***

**Code:**

#include<stdio.h>

int main()

{

    int row,i,j;

    printf("enter a number: ");

    scanf("*%d*",&row);

    for(i=1;i<=row;i++)

    {

        for(j=1;j<=i;j++)

        {

            printf("\* ");

            if(j == i)

            {

                printf("\n");

            }

        }

    }

}

**Output:**

**enter a number: 4  
\*   
\* \*  
\* \* \*  
\* \* \* \***

**Conclusion:**

**In conclusion, the provided C code generates a pattern of asterisks in the form of right-angled triangles based on user input. The user is prompted to enter a number, representing the number of rows in the pattern. The code then uses nested loops to iterate through each row and column, printing asterisks in a triangular pattern. The inner loop prints an asterisk and checks if the column number is equal to the row number, indicating the end of a row. If true, it adds a newline character to move to the next row.**

**This code is a basic implementation for creating a right-angled triangular pattern of asterisks and serves as an introductory example for understanding nested loops and simple pattern printing in C programming.**

**ii)**

**\***

**\* \***

**\* \* \***

**\* \* \* \***

**Code:**

#include<stdio.h>

int main()

{

    int a,j;

    printf("enter the number: ");

    scanf("*%d*",&a);

    for(int i=1;i<=a;i++)

    {

        for(j=1;j<=a-i;j++)

        {

            printf(" ");

        }

        for(int k=j;k<=a;k++)

        {

            printf("\*");

        }

        printf("\n");

    }

}

**Output:**

**enter the number: 4  
 \*  
 \*\*  
 \*\*\*  
\*\*\*\***

**Conclusion:**

**In conclusion, the provided C code generates a pattern of asterisks in the form of a right-angled triangle with the right-angle on the left side. The user is prompted to enter a number, which determines the height of the triangle. The code then utilizes a combination of nested loops to achieve the pattern. The outer loop controls the rows, the first inner loop prints leading spaces, and the second inner loop prints the asterisks. The number of asterisks in each row is determined by the current value of 'a' and the loop variables.**

**This code demonstrates a common approach to creating a left-aligned right-angled triangular pattern and is a useful example for understanding the manipulation of loops to generate specific patterns in C programming.**

**iii)**

**\***

**\* \***

**\* \* \***

**\* \* \* \***

**Code:**

#include<stdio.h>

int main()

{

    int a,j;

    printf("enter the number: ");

    scanf("*%d*",&a);

    for(int i=1;i<=a;i++)

    {

        for(j=1;j<=a-i;j++)

        {

            printf(" ");

        }

        for(int k=j;k<=a;k++)

        {

            printf("\* ");

        }

        printf("\n");

    }

}

**Output:**

**enter the number: 4  
 \*   
 \* \*  
 \* \* \*  
\* \* \* \***

**Conclusion:**

**In conclusion, the provided C code creates a pattern of asterisks in the shape of a right-angled triangle with the right angle on the left side. The user is prompted to input a number, determining the height of the triangle. The code employs nested loops to control the row and column printing. The first inner loop handles the leading spaces based on the row number, while the second inner loop prints asterisks followed by a space.**

**One noteworthy difference in this code compared to the previous version is that each asterisk is followed by a space, introducing spacing between the asterisks in each row. This modification enhances the visual appearance of the triangular pattern. Overall, the code serves as an illustrative example of how nested loops can be utilized to generate specific patterns in C programming.**

**Problem 5:**

**Write a program to convert temperature from Centigrade to Fahrenheit and vice-versa.**

**Code:**

#include<stdio.h>

float c\_to\_f(float *c*);

float f\_to\_c(float *f*);

int main()

{

    int choice;

    printf("enter which do you want to do: \n");

    printf("1 for C to F, 2 for F to C: \n");

    scanf("*%d*",&choice);

    if (choice == 1)

    {

        float C;

        printf("enter temperature in C: ");

        scanf("*%f*",&C);

        printf("*%f* Celsius to Fahrenheit is *%f*",C,c\_to\_f(C));

    }

    else if(choice == 2)

    {

        float F;

        printf("enter temperature in F: ");

        scanf("*%f*",&F);

        printf("*%f* Fahrenheit to Celsius is *%f*",F,f\_to\_c(F));

    }

    return 0;

}

float c\_to\_f(float *c*)

{

    float f;

    f = ((9\**c*)/5)+32;

    return f;

}

float f\_to\_c(float *f*)

{

    float c;

    c = ((*f*-32)/9)\*5;

    return c;

}

**Output:**

**Case 1:**

**enter which do you want to do:   
1 for C to F, 2 for F to C:   
1  
enter temperature in C: 100.34  
100.339996 Celsius to Fahrenheit is 212.612000**

**Case 2:**

**enter which do you want to do:   
1 for C to F, 2 for F to C:  
2  
enter temperature in F: 32.22  
32.220001 Fahrenheit to Celsius is 0.122223**

**Conclusion:**

**In conclusion, the provided C code is a simple temperature conversion program that allows users to convert temperatures between Celsius and Fahrenheit. The user is prompted to choose the conversion type: from Celsius to Fahrenheit (choice 1) or from Fahrenheit to Celsius (choice 2). Depending on the choice, the program then prompts the user to input the temperature in the specified unit and calculates and displays the converted temperature using the respective conversion functions.**

**The code includes separate functions for converting Celsius to Fahrenheit (`c\_to\_f`) and Fahrenheit to Celsius (`f\_to\_c`). These functions encapsulate the conversion logic, providing modularity to the code.**

**Overall, this code serves as a basic example of using functions to perform temperature conversions and demonstrates a simple menu-driven approach for user interaction in a C program.**

**Problem 6:**

**Write a program to generate a marksheet based on “Gradation” for a given marks**

**Code:**

#include<stdio.h>

#define AND &&

int main()

{

    int mark;

    printf("enter your obtained marks: ");

    scanf("*%d*",&mark);

    if(mark>=80 AND mark<=100)

        printf("your grade is AA");

    else if(mark<80 AND mark>=75)

        printf("your grade is A");

    else if(mark<75 AND mark>=70)

        printf("your grade is B");

    else if(mark<70 AND mark>=60)

        printf("your grade is C");

    else if(mark<60 AND mark>=0)

        printf("your grade is D");

    else if(mark>100)

        printf("Invalid input");

    else

        printf("Invalid input");

}

**Output:**

**Case 1:**

**enter your obtained marks: 50  
your grade is D**

**case 2:**

**enter your obtained marks: 60  
your grade is C**

**case 3:**

**enter your obtained marks: 70  
your grade is B**

**case 4:**

**enter your obtained marks: 80  
your grade is AA**

**case 5:**

**enter your obtained marks: -2  
Invalid input**

**case 6:**

**enter your obtained marks: 101  
Invalid input  
  
conclusion:**

**In conclusion, the provided C code is a simple program that determines the grade based on the user's obtained marks. The user is prompted to input their marks, and the program uses a series of if-else statements to evaluate the range of marks and assign the corresponding grade. The code defines a macro (`AND`) to represent the logical "and" operator for clarity.**

**The grading system is set up with different ranges for grades AA, A, B, C, and D. If the entered marks fall within a specific range, the program prints the corresponding grade. If the entered marks are out of the valid range (less than 0 or greater than 100), the program outputs an "Invalid input" message.**

**Overall, this code demonstrates a basic implementation of a grading system in C and serves as an example of conditional statements for decision-making based on user input.**

**Problem 7:**

**Write a program to find out the prime numbers within a given range.**

**Code:**

#define OR ||

#include<stdio.h>

#include<math.h>

int check\_prime(int);

int main()

{

    int n,a;

    printf("enter the range like(a b): ");

    scanf("*%d* *%d*",&a,&n);

    for(a;a<=n;a++)

    {

        if (check\_prime(a) == 0)

            printf("*%d* is a prime number\n",a);

*// else if(check\_prime(a) == 1)*

*//  printf("%d is not a prime number\n",a);*

        else if(check\_prime(a) == 2)

            printf("*%d* is nither a prime nor non-prime number\n",a);

        else if(check\_prime(a) == 3)

            printf("*%d* is a negative number\n",a);

    }

}

int check\_prime(int *n*)

{

    if(*n*<0)

        return 3; *// representing negative numbers*

    else if ((*n* == 0) OR (*n* == 1))

        return 2; *// neither prime nor non-prime*

    else

    {

        int count = 0;

        int range = sqrt(*n*) + 1;

        for(int i = 1;i<range;i++)

        {

            if (*n*%i == 0)

            {

                count += 1;

                if (count > 1)

                {

                return 1; *// for non-prime numbers*

                break;

                }

            }

        }

        if (count == 1)

        {

            return 0; *// for prime numbers*

        }

    }

}

**Output :**

**Case 1:**

**enter the range like(a b): 30 40  
31 is a prime number  
37 is a prime number**

**case 2:**

**enter the range like(a b): -2 0  
-2 is a negative number  
-1 is a negative number  
0 is nither a prime nor non-prime number  
  
conclusion:**

**In conclusion, the provided C code is designed to check and classify numbers within a specified range as prime, non-prime, or negative. The user is prompted to input a range (two integers), and the program then iterates through the numbers within that range. For each number, the `check\_prime` function is called to evaluate its primality.**

**The `check\_prime` function handles the classification logic. If a number is negative, it returns 3, representing a negative number. If a number is 0 or 1, it returns 2, indicating neither prime nor non-prime. For other positive numbers, the function checks for divisors up to the square root of the number to determine primality.**

**The main function then prints the classification result based on the returned values from `check\_prime`. The code effectively utilizes a macro (`OR`) to represent the logical "or" operator for improved readability.**

**In summary, this code serves as an illustrative example of a program that checks and categorizes numbers within a range based on their primality and negativity.**

**Problem 8:**

**Write a program to find out the factors of a given number.**

**Code:**

#include<stdio.h>

int main()

{

    int n;

    printf("enter the number: ");

    scanf("*%d*",&n);

    printf("factors of the number *%d* is\n",n);

    for(int i = 1;i<(n+1);i++)

    {

        if(n%i == 0)

        {

            printf("*%d* ",i);

        }

    }

}

**Output:**

**Case 1:**

**enter the number: 15  
factors of the number 15 is  
1 3 5 15**

**Case 2:**

**enter the number: 7  
factors of the number 7 is  
1 7**

**Case 3:**

**enter the number: 120  
factors of the number 120 is  
1 2 3 4 5 6 8 10 12 15 20 24 30 40 60 120**

**Conclusion:**

**In conclusion, the provided C code successfully identifies and prints the factors of a given number. However, there is a potential limitation when dealing with large input values that result in factorials exceeding the limit of the `int` datatype. The `int` datatype has a finite range, and when the computed factorial exceeds this range, the program may output incorrect values due to integer overflow.**

**To address this limitation, it would be beneficial to implement a more robust solution that can handle larger factorials. This could involve using a datatype with a larger range, such as `long long int` or `uint64\_t`, to accommodate larger factorials without encountering overflow issues. Additionally, the program could include checks to ensure that the input number is within a reasonable range to prevent unintended errors or inaccurate results due to limitations in the chosen datatype.**

**In summary, while the current code effectively finds factors, there is a consideration for potential issues with integer overflow for large factorials, and addressing this concern would contribute to the overall robustness of the program.**

**Problem 9:** **Write a program to calculate the factorial of a given number.**

**Code:**

#include<stdio.h>

int main()

{

    int n,fact = 1;

    printf("enter the number: ");

    scanf("*%d*",&n);

    if (n >= 0)

    {

        if (n == 0)

            fact = 1;

        else

        {

            for(int i = 1;i<=n;i++)

                fact \*= i;

        }

        printf("factorial of *%d* is = *%d*",n,fact);

    }

    else if(n < 0)

        printf("not defined");

    return 0;

}

**Output:**

**Case 1:**

**enter the number: 0  
factorial of 0 is = 1**

**case 2:**

**enter the number: 5  
factorial of 5 is = 120**

**case 3:  
enter the number: -5  
not defined**

**Conclusion:**

**In conclusion, the provided C code efficiently calculates and prints the factorial of a given non-negative integer. The program prompts the user to input a number (`n`), and based on the input, it uses a conditional structure to handle different scenarios. If `n` is greater than or equal to zero, the code employs a `for` loop to iteratively compute the factorial. In the case of `n` being zero, the factorial is set to 1. If `n` is negative, the program outputs "not defined" since factorials are not defined for negative numbers.**

**The code demonstrates a well-structured approach to handling various conditions and provides a clear output of the calculated factorial for valid input. It effectively addresses the issue of negative input values by explicitly stating that factorials are not defined in such cases. Overall, this code serves as a concise and functional solution for computing factorials in C.**

**Problem 10:** **Write a program to generate the Fibonacci series up to Nth term.**

**Code:**

#include<stdio.h>

int main()

{

    int n,a,b,i,z,t;

    a = 0;b = 1;

    printf("enter the n th number: ");

    scanf("*%d*",&n);

    for(i = 0;i<n;i++)

    {

        if((i==0)||(i==1))

        {

            printf("*%d* ",i);

        }

        else

        {

            z = a+b;

            t = b;

            b = z;

            a = t;

            printf("*%d* ",z);

        }

    }

    return 0;

}

**Output:**

**enter the n th number: 6  
0 1 1 2 3 5**

**Conclusion:**

**In conclusion, the provided C code generates and prints the Fibonacci series up to the nth term. The user is prompted to input the value of 'n,' and the program uses a `for` loop to iteratively calculate and display the Fibonacci sequence. The code initializes the first two terms (`a` and `b`) to 0 and 1, respectively.**

**The loop then follows a specific logic: for the first two terms, it prints the term itself. For subsequent terms, it calculates the next Fibonacci number (`z`) as the sum of the previous two terms (`a` and `b`). The variables are then updated to prepare for the next iteration.**

**This code effectively implements the Fibonacci sequence generation using a straightforward iterative approach. It serves as a clear example of how to calculate and display Fibonacci numbers up to a specified term in C programming.**

**Problem 11:** **Write a menu-driven program to implement different arithmetic operations.**

**Code:**

#include<stdio.h>

int main()

{

    int choice;

    printf("enter your choice: \n");

    printf("1 for addition\n");

    printf("2 subtraction\n");

    printf("3 multiplication\n");

    printf("4 division\n");

    scanf("*%d*",&choice);

    float a,b;

    printf("first num: ");

    scanf("*%f*",&a);

    printf("second num: ");

    scanf("*%f*",&b);

    if (choice == 1)

    {

        printf("*%f* + *%f* = *%f*",a,b,a+b);

    }

    else if(choice == 2)

    {

        printf("*%f* - *%f* = *%f*",a,b,a-b);

    }

    else if(choice == 3)

    {

        printf("*%f* x *%f* = *%f*",a,b,a\*b);

    }

    else if(choice == 4)

    {

        if (b == 0)

            printf("division by Zero not possible!!");

        else

            printf("*%f* / *%f* = *%f*",a,b,a/b);

    }

    else

        printf("invalid input");

}

**Output:**

**Case 1:**

**enter your choice:  
1 for addition  
2 substraction  
3 multiplication  
4 division  
1  
first num: 34  
34.000000 + 23.000000 = 57.000000**

**case 2:**

**enter your choice:  
1 for addition  
2 substraction  
3 multiplication  
4 division  
2  
first num: 45  
second num: 23  
45.000000 - 23.000000 = 22.000000**

**case 3:**

**enter your choice:  
1 for addition  
2 substraction  
3 multiplication  
4 division  
3  
first num: 23  
second num: 5  
23.000000 x 5.000000 = 115.000000**

**case 4:**

**enter your choice:  
1 for addition  
2 substraction  
3 multiplication  
4 division  
4  
first num: 34  
second num: 2  
34.000000 / 2.000000 = 17.000000**

**case 5:**

**enter your choice:   
1 for addition  
2 substraction  
3 multiplication  
4 division  
4  
first num: 23  
second num: 0  
division by Zero not possible!!**

**Conclusion:**

**In conclusion, the provided C code is a simple calculator program that performs basic arithmetic operations based on user input. The user is prompted to choose an operation (addition, subtraction, multiplication, or division) and then input two numbers. The code utilizes a series of conditional statements to execute the chosen operation and display the result.**

**The implementation includes checks to handle specific scenarios, such as division by zero. If the user selects division and the second number is zero, the program correctly outputs a message indicating that division by zero is not possible.**

**Overall, this code serves as a clear example of a basic calculator program in C, showcasing the use of conditional statements for different arithmetic operations and including error handling for certain cases.**

**Problem 12:** **Write a program to print a number in word.**

**Code:**

#include<stdio.h>

void num\_to\_word(int *a*)

{

    char \*one\_units[] = {"zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};

char \*two\_one\_units[] = {"ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"};

    char \*two\_units[] = {"", "", "twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty", "ninety"};

    if(*a*<10)

    {

        printf("*%s*",one\_units[*a*]);

    }

    else if(*a*<100)

    {

        if(*a*>=10 && *a*<20)

        {

            printf("*%s*",two\_one\_units[*a*-10]);

        }

        else

        {

            int sec\_unit = *a*/10;

            printf("*%s* ",two\_units[sec\_unit]);

            int rem = *a*%10;

                if(rem!=0)

                {

                    printf("*%s*",one\_units[rem]);

                }

        }

    }

    else if(*a*<1000)

    {

        int thr\_unit = *a*/100;

        printf("*%s* hundred ",one\_units[thr\_unit]);

        int rem = *a*%100;

            if(rem!=0)

            {

                if(rem>=10 && rem<20)

                {

                    printf("*%s* ",two\_one\_units[rem-10]);

                }

                else

                {

                    int sec\_unit = rem/10;

                    printf("*%s* ",two\_units[sec\_unit]);

                    int rem\_2 = rem%10;

                    if(rem\_2!=0)

                    {

                        printf("*%s* ",one\_units[rem\_2]);

                    }

                }

            }

    }

    else if(*a*<10000)

    {

        int four\_unit = *a*/1000;

        printf("*%s* thousand ",one\_units[four\_unit]);

        int rem = *a*%1000;

            if(rem != 0)

            {

                int thr\_unit = rem/100;

                printf("*%s* hundred ",one\_units[thr\_unit]);

                int rem\_2 = rem%100;

                    if(rem\_2!=0)

                    {

                        if(rem\_2>=10 && rem\_2<20)

                        {

                            printf("*%s* ",two\_one\_units[rem\_2-10]);

                        }

                        else

                        {

                            int sec\_unit = rem\_2/10;

                            printf("*%s* ",two\_units[sec\_unit]);

                            int rem\_3 = rem\_2%10;

                            if(rem\_3!=0)

                            {

                                printf("*%s* ",one\_units[rem\_3]);

                            }

                        }

                    }

            }

    }

    else if(*a*<100000)

    {

        int lst\_unit = *a*/1000;

        if(lst\_unit>=10 && lst\_unit<20)

        {

            printf("*%s*",two\_one\_units[lst\_unit-10]);

        }

        else

        {

            int sec\_unit = lst\_unit/10;

            printf("*%s* ",two\_units[sec\_unit]);

            int rem = lst\_unit%10;

                if(rem!=0)

                {

                    printf("*%s*",one\_units[rem]);

                }

        }

        printf(" thousand ");

        int rem = *a*%1000;

            if(rem != 0)

            {

                int thr\_unit = rem/100;

                printf("*%s* hundred ",one\_units[thr\_unit]);

                int rem\_2 = rem%100;

                    if(rem\_2!=0)

                    {

                        if(rem\_2>=10 && rem\_2<20)

                        {

                            printf("*%s* ",two\_one\_units[rem\_2-10]);

                        }

                        else

                        {

                            int sec\_unit = rem\_2/10;

                            printf("*%s* ",two\_units[sec\_unit]);

                            int rem\_3 = rem\_2%10;

                            if(rem\_3!=0)

                            {

                                printf("*%s* ",one\_units[rem\_3]);

                            }

                        }

                    }

            }

    }

}

int main()

{

    int a;

    printf("enter a number between(-100000 - 100000): ");

    scanf("*%d*",&a);

    if(a<0)

    {

        printf("minus ");num\_to\_word(-a);

    }

    else

        num\_to\_word(a);

}

**Output:**

**Case1:  
  
enter a number between(-100000 - 100000): 41  
forty one**

**Case2:**

**enter a number between(-100000 - 100000): -41  
minus forty one**

**Conclusion:**

**In conclusion, the provided C program efficiently converts an input integer, ranging from -100000 to 100000, into its word representation. The program utilizes arrays to store words for single digits, teens, tens, hundreds, and thousands. It handles both positive and negative numbers, offering a clear and concise word representation for each digit in the input. The code is organized and easy to understand, making it a functional tool for converting numerical values into words**

**Problem 13: Write a program to check a character is an alphabet, digit or special character.**

**Code:**

#include<stdio.h>

#define AND &&

#define OR ||

int main()

{

    char a;

    printf("enter the character: ");

    scanf("*%c*",&a);

    if((a >= 33 AND a<= 47) OR (a >= 58 AND a<= 64) OR (a >= 91 AND a<= 96) OR (a >= 123 AND a<= 126))

        printf("*%c* is a special character\n",a);

    else

        if (a >= 48 AND a <= 57)

            printf("*%c* is a digit\n",a);

        else

            if((a >= 65 AND a <= 90) OR (a>=97 AND a<=122))

                printf("*%c* is an alphabet\n",a);

            else

                printf("Unknown Character");

    return 0;

}

**Output:**

**Case 1:**

**enter the character: a  
a is an alphabet**

**case 2:**

**enter the character: !  
! is a special character**

**Case 3:**

**enter the character: 2  
2 is a digit**

**Conclusion:**

**In conclusion, the provided C code is a character classification program that determines whether a given input character falls into the categories of special characters, digits, alphabets, or an unknown character type. The user is prompted to enter a character, and the program uses a series of nested conditional statements to evaluate and classify the input character based on its ASCII value.**

**The code effectively employs the logical operators `AND` and `OR`, defined using the `#define` preprocessor directive, to create a comprehensive set of conditions for categorizing the character. It covers special characters, digits, and alphabets, providing clear output messages for each case.**

**Overall, this code serves as an illustrative example of how conditional statements can be used to categorize and process input characters in C programming.**

**Problem 14: Write a program to print the elements of an array in reverse order.**

**Code:**

#include<stdio.h>

int main()

{

    int n;

    printf("enter the number of elements in the array: ");

    scanf("*%d*",&n);

    char arr[n];

    for(int i = 0;i<n;i++)

    {

        printf("arr[*%d*] = ",i+1);

        scanf("*%d*",&arr[i]);

    }

    printf("reversed elements of the array is :\n");

    for(int i = (n-1);i>=0;i--)

    {

        printf("*%d* ",arr[i]);

    }

    return 0;

}

**Output:**

**enter the number of elements in the array: 5  
arr[1] = 34  
arr[2] = 2  
arr[3] = 65  
arr[4] = 87  
arr[5] = 23  
reversed elements of the array is :  
23 87 65 2 34**

**Conclusion:**

**In conclusion, the provided C code is designed to reverse the elements of an array based on user input. The program prompts the user to enter the number of elements for the array, dynamically allocates memory for the array using variable-length array (VLA) syntax, and then initializes each array element with user-input values. After populating the array, the program prints the reversed order of the elements.**

**The code effectively utilizes a `for` loop to collect user input and populate the array, followed by another `for` loop to iterate through the array in reverse order and print the reversed elements.**

**It's important to note that there's a small error in the code: the array `arr` is declared as a character array, but the `scanf` function is used with the `%d` format specifier, which is intended for integers. To resolve this, the format specifier in the `scanf` function should be changed to `%c` to match the character array type.**

**In summary, this code serves as a basic example of reversing elements in an array and showcases the use of loops for both user input and array manipulation in C programming.**

**Problem 15:** **Write a program to get the summation of all the numbers in an array.**

**Code:**

#include<stdio.h>

int main()

{

    int a;

    printf("enter the number of the elements in the array: ");

    scanf("*%d*",&a);

    char arr[a];

    for(int i = 0;i<a;i++)

    {

        printf("arr[*%d*] = ",i+1);

        scanf("*%d*",&arr[i]);

    }

    int sum = 0;

    for(int i = 0;i<a;i++)

    {

        sum += arr[i];

    }

    printf("total sum is *%d*",sum);

}

**Output:**

**enter the number of the elements in the array: 10  
arr[1] = 1  
arr[2] = 2  
arr[3] = 3  
arr[4] = 4  
arr[5] = 5  
arr[6] = 6  
arr[7] = 7  
arr[8] = 8  
arr[9] = 9  
arr[10] = 10  
total sum is 55  
  
conclusion:**

**In conclusion, the provided C code is designed to calculate the sum of elements in an array. The user is prompted to input the number of elements for the array, and the program dynamically allocates memory for the array using variable-length array (VLA) syntax. The code then proceeds to collect user input for each array element and calculates the sum of all elements using a `for` loop.**

**The program effectively utilizes loops for user input, array population, and sum calculation, providing a clear example of basic array manipulation in C programming. However, there is a small error in the code: the array `arr` is declared as a character array, but the `scanf` function is used with the `%d` format specifier, which is intended for integers. To resolve this, the format specifier in the `scanf` function should be changed to `%c` to match the character array type.**

**In summary, this code serves as a straightforward illustration of calculating the sum of elements in an array and demonstrates the use of loops and dynamic memory allocation in C.**

**Problem 16:** **Write a program to find out the duplicate elements in an array.**

**Code:**

#include<stdio.h>

void check(int \**i*,int \**j*,int *arr*[],int \**a*,int \**count*);

int main()

{

    int count =1,a;

    printf("enter the number of elements in the array: ");

    scanf("*%d*",&a);

    int arr[a];

    for(int i = 0;i<a;i++)

    {

        printf("arr[*%d*] = ",i+1);

        scanf("*%d*",&arr[i]);

    }

    for (int i = 0;i<a;i++)

    {

        for(int j = 0 ;j<a;j++)

        {

            if(i!=j)

            {

                check(&i,&j,arr,&a,&count);

            }

        }

        if(count != 1)

        {

        printf("duplicate element is *%d* \n",arr[i]);

        }

        count = 1;

    }

}

void check(int \**i*,int \**j*,int *arr*[],int \**a*,int \**count*)

{

    if(*arr*[\**i*] == *arr*[\**j*])

    {

        \**count* += 1;

        int k;

        for(k = \**j*;k<(\**a*-1);k++)

        {

*arr*[k] = *arr*[k+1];

        }

        \**a* -= 1;

        \**j* -= 1;

    }

}

**Output:**

**enter the number of elements in the array: 4  
arr[1] = 2  
arr[2] = 3  
arr[3] = 2  
arr[4] = 5  
duplicate element is 2**

**Conclusion:**

**In conclusion, the provided C code is designed to identify and display duplicate elements in an array. The user is prompted to input the number of elements for the array, and the program dynamically allocates memory for the array using variable-length array (VLA) syntax. The code then collects user input for each array element.**

**The main logic of the program involves nested loops to compare each element with every other element in the array. The `check` function is called to determine if there are duplicate elements, and if found, it adjusts the array to eliminate duplicates.**

**The program effectively identifies and displays duplicate elements, and it employs a separate function (`check`) for improved modularity. The use of pointers facilitates the manipulation of array elements and counters.**

**However, it's worth noting that the removal of duplicate elements in the array might alter its original order. Additionally, there is a small error in the code: the `scanf` function is used with `%d` format specifier for reading elements into an integer array, but `%d` is intended for integers, not for array elements. To address this, the format specifier should be changed to `%d` to correctly match the array element type.**

**In summary, this code serves as a practical example of identifying and handling duplicate elements in an array in C, showcasing the use of loops, functions, and pointers for array manipulation.**

**Problem 17:** **Write a program to find out an element in an array.**

**Code:**

#include<stdio.h>

int main()

{

    int a,n,count = 0;

    printf("enter the number of the elements in the array: ");

    scanf("*%d*",&a);

    char arr[a];

    for(int i = 0;i<a;i++)

    {

        printf("arr[*%d*] = ",i+1);

        scanf("*%d*",&arr[i]);

    }

    printf("enter the element to find: ");

    scanf("*%d*",&n);

    for(int i = 0;i<a;i++)

    {

        if (arr[i]==n)

        {

            count++;

            printf("the element is found!!\n");

            printf("index number is *%d*\n",i+1);

        }

    }

    if(count==0)

        printf("there is no existance of this element in the array!!");

}

**Output:**

**Case 1:**

**enter the number of the elements in the array: 7  
arr[1] = 1  
arr[2] = 2  
arr[3] = 3  
arr[4] = 4  
arr[5] = 5  
arr[6] = 6  
arr[7] = 7  
enter the element to find: 3  
the element is found!!  
index number is 3**

**case 2:**

**arr[1] = 2  
arr[2] = 3  
arr[3] = 6  
arr[4] = 2  
arr[5] = 7  
arr[6] = 7  
arr[7] = 4  
enter the element to find: 9  
there is no existance of this element in the array!!  
  
conclusion:**

**In conclusion, the provided C code is a program that searches for a specific element in an array. The user is prompted to input the number of elements for the array, and the program dynamically allocates memory for the array using variable-length array (VLA) syntax. The code then collects user input for each array element.**

**Subsequently, the user is prompted to input an element to search for within the array. The program uses a `for` loop to iterate through the array and checks if the input element matches any of the elements in the array. If a match is found, it prints a message indicating the presence of the element, along with its index. If no match is found, it outputs a message indicating the absence of the element in the array.**

**The code effectively handles the search operation and provides clear output messages based on the search result. However, there is a small error in the code: the array `arr` is declared as a character array, but the `scanf` function is used with the `%d` format specifier, which is intended for integers. To resolve this, the format specifier in the `scanf` function should be changed to `%c` to match the character array type.**

**In summary, this code serves as a basic example of searching for an element in an array in C, showcasing the use of loops and user input processing.**

**Problem 18:** **Write a program to sort elements of an array in ascending and descending order.**

**Code:**

#include<stdio.h>

void swap(int \**a*,int \**b*); *// to swap to numbers*

void printarr(int *arr*[],int *a*); *// to print array elemnts*

int length(int *arr*[],int *size*);

int main()

{

    int a,count,c;

    printf("number of elements in the array: ");

    scanf("*%d*",&a);

    int arr[a];

    for(int i = 0;i<a;i++)

    {

        printf("arr[*%d*] :",i+1);

        scanf("*%d*",&arr[i]);

    }

    printf("\n");

    printf("enter 1 for ascending, or 2 for descending:  ");

    scanf("*%d*",&c);

    if (c == 1)

    {

        do{

            count = 0;

            for(int i = 0;i<a-1;i++)

            {

                if(arr[i]>arr[i+1])

                {

                    swap(&arr[i],&arr[i+1]);

                    count++;

                }

            }

        }while(count != 0);

        printarr(arr,a);

    }

    else if(c == 2)

    {

        do{

            count = 0;

            for(int i = 0;i<a-1;i++)

            {

                if(arr[i]<arr[i+1])

                {

                    swap(&arr[i],&arr[i+1]);

                    count++;

                }

            }

        }while(count != 0);

        printarr(arr,a);

    }

    else

    {

        printf("wrong input!!");

    }

}

void swap(int \**a*,int \**b*)

{

    int t;

    t = \**a*;

    \**a* = \**b*;

    \**b* = t;

}

void printarr(int *arr*[],int *a*)

{

    printf("{");

    for(int i = 0;i<*a*;i++)

    {

        printf("*%d*,",*arr*[i]);

    }

    printf("}");

}

int length(int *arr*[],int *size*)

{

    return *size*;

}

**Output:**

**Case 1:**

**number of elements in the array: 5  
arr[1] :12  
arr[2] :23  
arr[3] :34  
arr[4] :45  
arr[5] :56  
  
enter 1 for ascending, or 2 for descending: 1  
{12,23,34,45,56,}**

**case 2:**

**number of elements in the array: 5  
arr[1] :12  
arr[2] :23  
arr[3] :34  
arr[4] :45  
arr[5] :56  
  
enter 1 for ascending, or 2 for descending: 2  
{56,45,34,23,12,}**

**Problem 19:** **Write a program to count the frequency of each element in an array.**

**Code:**

#include<stdio.h>

int len(int *arr*[],int *size*);

void check(int \**i*,int \**j*,int *arr*[],int \**a*,int \**count*);

int main()

{

    int count =1;

    int arr[]={67,98,67,67,34,34};

    int a = len(arr,sizeof(arr)/sizeof(arr[0]));

    for (int i = 0;i<a;i++)

    {

        for(int j = 0 ;j<a;j++)

        {

            if(i!=j)

            {

                check(&i,&j,arr,&a,&count);

            }

        }

        printf("*%d* present there *%d* times\n",arr[i],count);

        count = 1;

    }

}

void check(int \**i*,int \**j*,int *arr*[],int \**a*,int \**count*)

{

    if(*arr*[\**i*] == *arr*[\**j*])

    {

        \**count* += 1;

        int k;

        for(k = \**j*;k<(\**a*-1);k++)

        {

*arr*[k] = *arr*[k+1];

        }

        \**a* -= 1;

        \**j* -= 1;

    }

}

int len(int *arr*[],int *size*)

{

    return *size*;

}

**Output:**

**Case 1:**

**number of elements in the array: 5  
arr[1] :1  
arr[2] :1  
arr[3] :1  
arr[4] :1  
arr[5] :1  
1 present there 5 times**

**Case 2:**

**number of elements in the array: 5   
arr[1] :1  
arr[2] :2  
arr[3] :1  
arr[4] :1  
arr[5] :2  
1 present there 3 times  
2 present there 2 times**

**Case 3:**

**number of elements in the array: 5  
arr[1] :1  
arr[2] :2  
arr[3] :3  
arr[4] :4  
arr[5] :5  
1 present there 1 times  
2 present there 1 times  
3 present there 1 times  
4 present there 1 times  
5 present there 1 times**

**Problem 20:** **Write a menu-driven program to implement matrix-addition, subtraction and multiplication.**

**Code:**

#include<stdio.h>

#include<stdlib.h>

void addiiton()

{

    int r1,c1;

    int r2,c2;

    int r3,c3;

*// 1st matrix*

    printf("enter the dimension of the first matrix(a b): \n");

    scanf("*%d* *%d*",&r1,&c1);

    printf("enter the dimension of the second matrix(a b): \n");

    scanf("*%d* *%d*",&r2,&c2);

*// 1st matrix*

    if(r1 != r2 || c1 != c2)

    {

        printf("in this dimension matrix addition is not possible!!!\n");

    }

    else

    {

*// 1st matrix*

        int arr1[r1][c1];

        printf("enter the elements of the first array: \n");

        for(int i = 0;i<r1;i++)

        {

            for(int j = 0;j<c1;j++)

            {

                scanf("*%d*",&arr1[i][j]);

            }

        }

*// 2nd matrix*

        int arr2[r2][c2];

        printf("enter the elements of the second array: \n");

        for(int i = 0;i<r2;i++)

        {

            for(int j = 0;j<c2;j++)

            {

                scanf("*%d*",&arr2[i][j]);

            }

        }

        r3 = r1;c3 = c2;

        int arr3[r3][c3];

        for(int i = 0;i<r3;i++)

        {

            for(int j = 0;j<c3;j++)

            {

                arr3[i][j] = arr1[i][j]+arr2[i][j];

            }

        }

        printf("\nthe addition of two matrix is: \n");

        for(int i = 0;i<r3;i++)

        {

            for(int j = 0;j<c3;j++)

            {

                printf("*%d* ",arr3[i][j]);

            }

            printf("\n");

        }

    }

}

void subtraction()

{

    int r1,c1;

    int r2,c2;

    int r3,c3;

*// 1st matrix*

    printf("enter the dimension of the first matrix(a b): \n");

    scanf("*%d* *%d*",&r1,&c1);

    printf("enter the dimension of the second matrix(a b): \n");

    scanf("*%d* *%d*",&r2,&c2);

*// 1st matrix*

    if(r1 != r2 || c1 != c2)

    {

        printf("in this dimension matrix subtraction is not possible!!!\n");

    }

    else

    {

*// 1st matrix*

        int arr1[r1][c1];

        printf("enter the elements of the first array: \n");

        for(int i = 0;i<r1;i++)

        {

            for(int j = 0;j<c1;j++)

            {

                scanf("*%d*",&arr1[i][j]);

            }

        }

*// 2nd matrix*

        int arr2[r2][c2];

        printf("enter the elements of the second array: \n");

        for(int i = 0;i<r2;i++)

        {

            for(int j = 0;j<c2;j++)

            {

                scanf("*%d*",&arr2[i][j]);

            }

        }

        r3 = r1;c3 = c2;

        int arr3[r3][c3];

        for(int i = 0;i<r3;i++)

        {

            for(int j = 0;j<c3;j++)

            {

                arr3[i][j] = arr1[i][j]-arr2[i][j];

            }

        }

        printf("\nthe subtraction of two matrix is: \n");

        for(int i = 0;i<r3;i++)

        {

            for(int j = 0;j<c3;j++)

            {

                printf("*%d* ",arr3[i][j]);

            }

            printf("\n");

        }

    }

}

void multiplication()

{

    int r3,c3;

    int r1,c1;

    int r2,c2;

*// 1st matrix*

    printf("enter the dimension of the first matrix(a b): \n");

    scanf("*%d* *%d*",&r1,&c1);

*// 2nd matrix*

    printf("enter the dimension of the second matrix(a b): \n");

    scanf("*%d* *%d*",&r2,&c2);

    if(r2 != c1)

    {

        printf("in this dimension matrix addition is not possible!!!\n");

    }

    else

    {

*//1st matrix*

        int arr1[r1][c1];

        printf("enter the elements of the first array: \n");

        for(int i = 0;i<r1;i++)

        {

            for(int j = 0;j<c1;j++)

            {

                scanf("*%d*",&arr1[i][j]);

            }

        }

*//2nd matrix*

        int arr2[r2][c2];

        printf("enter the elements of the second array: \n");

        for(int i = 0;i<r2;i++)

        {

            for(int j = 0;j<c2;j++)

            {

                scanf("*%d*",&arr2[i][j]);

            }

        }

        r3 = r1;c3 = c2;

        int arr3[r3][c3];

        for(int i = 0;i<r3;i++)*//0 1*

        {

            for(int j = 0;j<c3;j++) *//0*

            {

                int sum = 0;

                for(int k = 0;k<c1;k++)*// 0*

                {

                    sum += arr1[i][k]\*arr2[k][j];

                }

                arr3[i][j] = sum;

            }

        }

        printf("\nthe multiplication of two matrix is: \n");

        for(int i = 0;i<r3;i++)

        {

            for(int j = 0;j<c3;j++)

            {

                printf("*%d* ",arr3[i][j]);

            }

            printf("\n");

        }

    }

}

int main()

{

    int choice;

    printf("\nMatrix Operations Menu:\n");

        printf("1. Matrix Addition\n");

        printf("2. Matrix Subtraction\n");

        printf("3. Matrix Multiplication\n");

        printf("4. Exit\n\n");

        printf("Enter your choice (1-4): ");

        scanf("*%d*", &choice);

    if(choice!=4)

    {

        if(choice == 1)

        {

            addiiton();

        }

        else if (choice == 2)

        {

            subtraction();

        }

        else if(choice == 3)

        {

            multiplication();

        }

    }

    else

        exit(1);

}

**21. Write a program to create a student database of N number of records. Each record consists of followings:**

**Student\_Name, Student\_Roll, Total\_Marks, Subject  
(Ex: Amit Dey, 13, 345, Computer Science) Arrange the records with respect to Student\_Name, Student\_Roll and Total\_Marks respectively. Create the database using “structure” as well as “union” respectively.**

**Code:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

typedef struct student{

    char name[20];

    int roll;

    int total\_marks;

    char subject[20];

}student;

int main()

{

    int a;

    printf("how many numbers of records do you want to save: \n");

    scanf("*%d*",&a);

    student data[a];

    if(a>0)

    {

        for(int i = 0;i<a;i++)

        {

            printf("record no *%d*\n",i+1);

            printf("student\'s name: ");

            scanf(" %[^\n]",data[i].name);

            printf("student\'s roll: ");

            scanf("*%d*",&data[i].roll);

            printf("total marks of the student: ");

            scanf("*%d*",&data[i].total\_marks);

            printf("subject of the student: ");

            scanf(" %[^\n]",data[i].subject);

            printf("\n");

        }

        printf("\n");

        int choice;

        do

        {

            printf("how you want to arrange the database: \n1 -> name wise\n2 -> roll wise\n3 -> total marks wise\n4 -> exit:\n");

            scanf("*%d*",&choice);

            if(choice == 4)

            {

                exit(1);

            }

            else if(choice == 1)

            {

                for(int i = 0;i<a;i++)

                {

                    for(int j = i+1;j<a;j++)

                    {

                        if(strcmp(data[i].name,data[j].name) > 0)

                        {

                            student temp;

                            temp = data[i];

                            data[i] = data[j];

                            data[j] = temp;

                        }

                    }

                }

                printf("\n");

                printf("arranged by students name!\n");

            }

            else if(choice == 2)

            {

                for(int i = 0;i<a;i++)

                {

                    for(int j = i+1;j<a;j++)

                    {

                        if(data[i].roll > data[j].roll)

                        {

                            student temp;

                            temp = data[i];

                            data[i] = data[j];

                            data[j] = temp;

                        }

                    }

                }

                printf("\n");

                printf("arranged by students roll!\n");

            }

            else if(choice == 3)

            {

                for(int i = 0;i<a;i++)

                {

                    for(int j = i+1;j<a;j++)

                    {

                        if(data[i].total\_marks < data[j].total\_marks)

                        {

                            student temp;

                            temp = data[i];

                            data[i] = data[j];

                            data[j] = temp;

                        }

                    }

                }

                printf("\n");

                printf("arranged by students total marks!\n");

            }

            for(int i = 0;i<a;i++)

            {

                printf("student\'s name: *%s*\n",data[i].name);

                printf("student\'s roll: *%d*\n",data[i].roll);

                printf("total marks of the student: *%d*\n",data[i].total\_marks);

                printf("subject of the student: *%s*\n",data[i].subject);

                printf("\n");

            }

        }while(choice != 4);

    }

}

1. **Write a program to check a string (case sensitive) is palindrome or not.**

**Code:**

#include<stdio.h>

#include<string.h>

void reverse(char *arr1*[],char *arr2*[])

{

    int length = strlen(*arr1*);

    int i;

    for(i = 0;i != length;i++)

    {

*arr2*[i] = *arr1*[length-i-1];

    }

*arr2*[i] = *arr1*[length];

}

int is\_palindrome(char *string*[],char *rev\_string*[])

{

    int i;

    for(i=0;*string*[i]!='\0';i++)

    {

        if (*string*[i] != *rev\_string*[i])

        {

            return 0;

            break;

        }

    }

    return 1;

}

int main()

{

    char string[100];

    char rev\_string[100];

    int check;

    printf("enter the string: ");

    gets(string);

    int length = strlen(string);

    reverse(string,rev\_string);

    check = is\_palindrome(string,rev\_string);

    if (check == 1)

    {

        printf("*%s* is palindrome",string);

    }

    else if(check == 0)

    {

        printf("*%s* is not palindrome",string);

    }

}

1. **Write a program to sort a list of strings in ascending and descending order.**
2. **Write a program to read content of a “.txt” file and copy the content of the said file into another “.txt” file.**

**Code:**

#include<stdio.h>

int main()

{

    FILE \* fp;

    FILE \* fs;

    char ch;

//create two .txt file named new\_1.txt , new\_2.txt

    fp = fopen("new\_1.txt","r");

    fs = fopen("new\_2.txt","w");

    if (fp == NULL || fs == NULL)

        printf("file is not opend");

    while(1)

    {

        ch = fgetc(fp);

        if (ch == EOF)

            break;

        else

            fputc(ch,fs);

    }

    fclose(fp);fclose(fs);

}