Unit No-II

Control Structures

Relational operators are the symbols that are used for comparison between two values to understand the type of relationship a pair of numbers shares. The result that we get after the relational operation is a boolean value, that tells whether the comparison is true or false. Relational operators are mainly used in conditional statements and loops to check the conditions in C programming.

Operator	Description	Example
= =	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	if (A = = B) { }
! =	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	if (A != B) { }
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	if (A > B) { }
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	if (A < B) { }
< =	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	if (A >= B) { }
< =	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	if (A <= B) { }

Program:

```
// C program to demonstrate working of relational operators
#include <stdio.h>
int main()
{
     int a = 10, b = 4;
     // greater than example
     if (a > b)
          printf("a is greater than b\n");
     else
          printf("a is less than or equal to b\n");
     // greater than equal to
     if (a >= b)
          printf("a is greater than or equal to b\n");
     else
          printf("a is lesser than b\n");
```

```
    // less than example
    if (a < b)
        printf("a is less than b\n");
    else
        printf("a is greater than or equal to b\n");
    // lesser than equal to
    if (a <= b)
        printf("a is lesser than or equal to b\n");
    else
        printf("a is greater than b\n");
    // equal to
    if (a == b)
        printf("a is equal to b\n");
    else
        printf("a and b are not equal\n");
    // not equal to
    if (a != b)
        printf("a is not equal to b\n");
    else
        printf("a is equal b\n");
    return 0;
}
```

// C program to find the maximum number out of the three

```
// given numbers using if-else statement
#include <stdio.h>
int main()
{
    int A, B, C;
    printf("Enter the numbers A, B and C: ");
    scanf("%d %d %d", &A, &B, &C);
    // finding max using compound expressions
    if (A >= B && A >= C)
        printf("%d is the largest number.", A);
    else if (B >= A && B >= C)
        printf("%d is the largest number.", B);
    else
        printf("%d is the largest number.", C);
    return 0;
}
```

Logical Operator:

In C programming, logical operators are used to evaluate the truth value of an expression or a condition. They allow programmers to combine multiple conditions and test them

simultaneously. Logical operators are essential in programming, as they help to control program flow and determine the outcome of different operations.
There are three logical operators in C: "&&" (logical AND), "||" (logical OR), and "!" (logical NOT).

1. Logical AND Operator

The "&&" operator tests whether two conditions are true simultaneously. It returns true if both conditions are true, and false otherwise. For example, consider the following code:

```
#include <stdio.h>
int main()
{
int x = 22, y = 33;
if (x > 0 && y > 0)
{
    printf("Both values are greater than 0\n");
}
else
{
    printf("Both values are less than 0\n");
}
return 0;
}
```

Output

```
Both values are greater than 0
```

In this example, the "&&" operator tests whether both x and y are greater than zero. Since both conditions are true, the message "Both x and y are positive" will be printed to the console.

2. Logical OR Operator

The "||" operator tests whether at least one of two conditions is true. It returns true if either of the conditions is true, and false if both are false. For example:

```
#include <stdio.h>
int main()
{
int x = -2, y = 16;
if (x > 0 || y > 0)
{
    printf("Any of the given values is larger than 0\n");
}
else
{
    printf("Both values are less than 0\n");
}
return 0;
}
Output
Any of the given values is larger than 0
```

In this example, the "||" operator tests whether either x or y is greater than zero. Since y is greater than zero, the message " Any of the given values is larger than 0" will be printed to the console.

3. Logical NOT Operator

The "!" operator, also known as the logical NOT operator, reverses the truth value of a condition. It returns true if the condition is false, and false if the condition is true. For example:

```
#include <stdio.h>
#include <math.h>
int main ()
{
    int p = 8;
    printf (" The return value = %d \n", ! (p == 8));
    printf (" The return value = %d \n", ! (p != 8));
    printf (" The return value = %d \n", ! (p >= 4));
    printf (" The return value = %d \n", ! (p < 4));

    return 0;
}
```

Output

```
The return value = 0
 The return value = 1
 The return value = 0
 The return value = 1
```

In this example, the "!" operator is used to show the utilization of NOT operator.

if statement:

The if in C is a decision-making statement that is used to execute a block of code based on the value of the given expression. It is one of the core concepts of C programming and is used to include conditional code in program.

Syntax of if Statement in C

```
if(condition)
{
   // if body
   // Statements to execute if condition is true
}
```

// C Program to demonstrate the syntax of if statement

```
#include <stdio.h>
int main()
{
     int gfg = 9;
     // if statement with true condition
     if (gfg < 10) {
          printf("%d is less than 10", gfg);
```

```
	}
	// if statement with false condition
	if (gfg > 20) {
		printf("%d is greater than 20", gfg);
	}
	return 0;
}
```

Output

```
9 is less than 10
```

if-else Statement

The if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the specified condition (test expression). If the given condition is true, then the code inside the if block is executed, otherwise the code inside the else block is executed.

Syntax of if-else

```
if (condition)
{
    // code executed when the condition is true
}
else
{
    // code executed when the condition is false
}
```

// C Program to demonstrate the use of if-else statement

```
#include <stdio.h>
int main()
{
	// if block with condition at the start
	if (5 < 10) {
		// will be executed if the condition is true
		printf("5 is less than 10.");
	}
	// else block after the if block
	else {
		// will be executed if the condition is false
		printf("5 is greater that 10.");
	}
	return 0;
}
```

// C Program to Demonstrate the working of if-else statement

```
#include <stdio.h>
int main()
{
	// Some random number
	int num = 9911234;
	// checking the condition at the start of if block
	if (num % 2 == 0) {
```

```
          // executed when the number is even
          printf("Number is even");
     }
     // else block
     else {
          // executed when the number is odd
          printf("Number is Odd");
     }
     return 0;
}
```

// C Program to check whether the person is eligible to vote or not

```
#include <stdio.h>
int main()
{
     // declaring age of two person
     int p1_age = 15;
     int p2_age = 25;
     // checking eligibility of person 1
     if (p1_age < 18)
          printf("Person 1 is not eligible to vote.\n");
     else
          printf("Person 1 is eligible to vote.\n");
     // checking eligiblity of person 2
     if (p2_age < 18)
          printf("Person 2 is not eligible to vote.\n");
     else
          printf("Person 2 is eligible to vote.");
     return 0;
}
```

Nested if-else Statement

When a series of decision is required, nested if-else is used. Nesting means using one if-else construct within another one. If the condition in the outer if, is true, then only the inner if-else will get executed. Further the statements in the inner if will get execute only if the condition of inner if, evaluates to true. If it is false, the statements in inner else will get executed. If the outer if evaluates to false, then the statements in outer else get executed.

```
General syntax:
if(condition)
{
     if(condition)
     {
          statements
     }
     else
     {
          statements
     }
}
else
{
     statements
}
Statements
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int val;
     clrscr();
     printf("Enter a number");
     scanf("%d",&val);
     if(val>=5)
     {
          if(val>5)
          {
               printf("Number is greater than 5");
          }
          else
          {
               printf("Number is equal to 5");
          }
     }
     else
     {
          printf("Number is less than 5");
     }
getch();
}
```

if-else if Ladder

if else if ladder in C programming is used to test a series of conditions sequentially. Furthermore, if a condition is tested only when all previous if conditions in the if-else ladder are false. If any of the conditional expressions evaluate to be true, the appropriate code block will be executed, and the entire if-else ladder will be terminated.

Syntax:

```
// any if-else ladder starts with an if statement only
if(condition)
{
}
else if(condition)
{
 // this else if will be executed when condition in if is false and
 // the condition of this else if is true
}
.... // once if-else ladder can have multiple else if
else
{ // at the end we put else
}
```

Example:

// C Program to Calculate Grade According to marks

```
// using the if else if ladder
#include <stdio.h>
int main()
{
	int marks = 91;
	if (marks <= 100 && marks >= 90)
		printf("A+ Grade");
	else if (marks < 90 && marks >= 80)
		printf("A Grade");
	else if (marks < 80 && marks >= 70)
		printf("B Grade");
	else if (marks < 70 && marks >= 60)
		printf("C Grade");
	else if (marks < 60 && marks >= 50)
		printf("D Grade");
	else
		printf("F Failed");
	return 0;
}
```

Output

```
A+ Grade
```

Switch case

Switch case statement evaluates a given expression and based on the evaluated value (matching a certain condition), it executes the statements associated with it.
Basically, it is used to perform different actions based on different conditions(cases).

- Switch case statements follow a selection-control mechanism and allow a value to change control of execution.
- They are a substitute for long if statements that compare a variable to several integral values.
- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

In C, the switch case statement is used for executing one condition from multiple conditions. It is similar to an if-else-if ladder.
The switch statement consists of conditional-based cases and a default case.

Syntax of switch Statement in C

```
switch(expression)
{
	case 1: statement_1;
	break;
	case 2: statement_2;
	break;
	.
	.
	.
	case n: statement_n;
	break;
```

```
    default: default_statement;
}
```

Rules of the switch case statement
Following are some of the rules that we need to follow while using the switch statement:
In a switch statement, the “case value” must be of “char” and “int” type.
There can be one or N number of cases.
The values in the case must be unique.
Each statement of the case can have a break statement. It is optional.
The default Statement is also optional.

// C program to print the day using switch

```
#include <stdio.h>
int main()
{
    int day = 2;
    printf("The day with number %d is ", day);
    switch (day)
    {
    case 1:
        printf("Monday");
        break;
    case 2:
        printf("Tuesday");
        break;
    case 3:
        printf("Wednesday");
        break;
    case 4:
        printf("Thursday");
        break;
    case 5:
        printf("Friday");
        break;
    case 6:
        printf("Saturday");
        break;
    case 7:
        printf("Sunday");
        break;
    default:
        printf("Invalid Input");
        break;
    }
    return 0;
}
```

Output

```
The day with number 2 is Tuesday
```

While Loop:

The **while Loop** is an entry-controlled loop in C programming language. This loop can be used to iterate a part of code while the given condition remains true.

Syntax

The while loop syntax is as follows:

```
while (test expression)
{
   // body consisting of multiple statements
Increment or decrement
}
```

While Loop Structure:

The while loop works by following a very structured top-down approach that can be divided into the following parts:

1. **Initialization:** In this step, we initialize the **loop variable** to some **initial value.** Initialization is not part of while loop syntax but it is essential when we are using some variable in the test expression

2. **Conditional Statement:** This is one of the most crucial steps as it decides whether the block in the while loop code will execute. The while loop body will be executed if and only the **test condition** defined in the conditional statement is **true.**

3. **Body:** It is the actual set of statements that will be executed till the specified condition is true. It is generally enclosed inside **{ } braces.**

4. **Updating:** It is an expression that **updates** the value of the **loop variable** in each iteration. It is also not part of the syntax but we have to define it explicitly in the body of the loop.

Example:

```
// Print numbers from 1 to 5
#include <stdio.h>
int main()
{
     int i = 1;
     while (i <= 5)
     {
          printf("%d\n", i);
          ++i;
     }
return 0;
}
```

Output:

```
1
2
3
4
5
```

Do while loop:

A ***"do-while" loop*** is a form of a ***loop*** in C that executes the code block first, followed by the condition. If the condition is ***true***, the ***loop*** continues to run; else, it stops. However, whether the condition is originally ***true***, it ensures that the code block is performed at least once.

Do While Loop Syntax

```
Do
{
     //code to be executed
     Increment or decrement
}while(condition);
```

- The ***do keyword*** marks the beginning of the Loop.
- The ***code block*** within ***curly braces {}*** is the body of the loop, which contains the code you want to repeat.
- The ***while keyword*** is followed by a condition enclosed in parentheses (). After the code block has been run, this condition is verified. If the condition is ***true***, the loop continues else, the ***loop ends***.

Example:

Here is a simple example of a ***"do-while" loop*** in C that prints numbers from 1 to 5:

```
#include <stdio.h>
int main()
{
     inti = 1;
     do
     {
          printf("%d\n", i);
          i++;
     } while (i<= 5);
return 0;
}
```

Output:

```
1
2
3
4
5
```

Program to print table for the given number using do while Loop

```
#include<stdio.h>
int main()
```

```
{
     inti=1,number=0;
     printf("Enter a number: ");
     scanf("%d",&number);
     do
     {
          printf("%d \n",(number*i));
          i++;
     }while(i<=10);
return 0;
}
```

Output:

```
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
```

Let's take a program that prints the multiplication table of a given number N using a *do...while Loop*:

```
#include <stdio.h>
int main()
{
     int N;
     printf("Enter a number to generate its multiplication
     table: ");
     scanf("%d", &N);
     inti = 1;
     do
     {
          printf("%d x %d = %d\n", N, i, N * i);
          i++;
     } while (i<= 10);
   return 0;
}
```

Output:

```
Please enter a number to generate its multiplication table: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
```

```
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

while Loop	do…while Loop
The test condition is checked **before the loop body is executed.**	The test condition is checked **after executing the body.**
When the condition is false, the **body is not executed** not even once.	The body of the **do…while loop is executed at least once** even when the condition is false.
It is a type of **pre-tested or entry-controlled loop.**	It is a type of **post-tested or exit-controlled loop.**
Semicolon is not required.	Semicolon is required at the end.

For Loop:
The **for loop** in C Language provides a functionality/feature to repeat a set of statements a defined number of times. The for loop is in itself a form of an **entry-controlled loop**.
Unlike the while loop and do…while loop, the for loop contains the initialization, condition, and updating statements as part of its syntax. It is mainly used to traverse arrays, vectors, and other data structures.

Syntax of for Loop

```
for(initialization; Condition; Increment/decrement)
{
     // body consisting of multiple statements
}
```

Structure of for Loop

The for loop follows a very structured approach where it begins with initializing a condition then checks the condition and, in the end, executes conditional statements followed by an updating of values.

1. **Initialization:** This step initializes a loop control variable with an initial value that helps to progress the loop or helps in checking the condition. It acts as the index value when iterating an array or string.

2. **Condition:** This step of the **for loop** defines the condition that determines whether the loop should continue executing or not. The condition is checked before each iteration and if it is true then the iteration of the loop continues otherwise the loop is terminated.

3. **Body:** It is the set of statements i.e. variables, functions, etc that is executed repeatedly till the condition is true. It is enclosed within curly braces **{ }**.

4. **Increment/decrement:** This specifies how the loop control variable should be updated after each iteration of the loop. Generally, it is the incrementation (variable++) or decrementation (variable–) of the loop control variable.

```
Example
#include <stdio.h>
int main()
{
  int i;
    for (i = 1; i <= 10; i++)
     {
          printf("%d ", i);
     }
    printf("\nThis statement executes after for loop end!!!!");
 return 0;
}
```

Advantages of for Loop

There are certain advantages of using for loops in C as mentioned below:

- Provides code reusability
- Code size decreases
- Traversing in data structures like array and string becomes easy.

Disadvantages of for Loop

- Can’t skip any element while traversing
- Only a single condition is followed

Branching Statement:

Break keyword:

The **break in C** is a loop control statement that breaks out of the loop when encountered. It can be used inside loops or switch statements to bring the control out of the block. The break statement can only break out of a single loop at a time.

Syntax of break in C

break;

Use of break in C

The break statement in C is used for breaking out of the loop. We can use it with any type of loop to bring the program control out of the loop. In C, we can use the break statement in the following ways:

- **Simple Loops**
- **Nested Loops**
- **Infinite Loops**
- **Switch case**

Simple Loops

Break statements in C can be used with simple loops i.e, for loops, while loops, and do-while loops.

```
// C Program to demonstrate break statement with for loop
#include <stdio.h>
int main()
{
     // using break inside for loop to terminate after 2
     // iteration
     printf("break in for loop\n");
     for (int i = 1; i < 5; i++)
     {
          if (i == 3)
          {
               break;
          }
          else
          {
               printf("%d ", i);
          }
     }
     // using break inside while loop to terminate after 2
     // iteration
     printf("\nbreak in while loop\n");
     int i = 1;
     while (i < 20) {
          if (i == 3)
               break;
          else
               printf("%d ", i);
          i++;
     }
     return 0;
}
```

Output

```
break in for loop
1 2
break in while loop
1 2
```

Nested Loops

Break statements can also be used when working with nested loops. The control will come out of only that loop in which the break statement is used.

```
// program using break statement in Nested loops
#include <stdio.h>
```

```
int main()
{
	// nested for loops with break statement
	// at inner loop
	for (int i = 1; i <= 6; ++i)
	{
		for (int j = 1; j <= i; ++j)
		{
			if (i <= 4)
			{
				printf("%d ", j);
			}
			else
			{
				// if i > 4 then this innermost loop will
				// break
				break;
			}
		}
		printf("\n");
	}
	return 0;
}
```

Output

```
1

1 2

1 2 3

1 2 3 4
```

Note: Break statement only breaks out of one loop at a time. So if in nested loop, we have used break in inner loop, the control will come to outer loop instead of breaking out of all the loops at once. We will have to use multiple break statements if we want to break out of all the loops.

Infinite Loops

An Infinite loop can be terminated with a break statement as a part of the condition.

```
// C Program to demonstrate infinite loop without using
// break statement
#include <stdio.h>
int main()
{
	int i = 0;
```

```
    // while loop which will always be true
    while (1)
    {
        printf("%d ", i);
        i++;
        if (i == 5)
        {
            break;
        }
    }
    return 0;
}
```

Break in C switch case

In general, the Switch case statement evaluates an expression, and depending on the value of the expression, it executes the statement associated with the value. Not only that, all the cases after the matching case after the matching case will also be executed. To prevent that, we can use the break statement in the switch case as shown:

```
// C Program to demonstrate working of break with switch case
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char c;
    float x, y;
    while (1) {
        printf("Enter an operator (+, -), if want to exit "
            "press x: ");
        scanf(" %c", &c);
        // to exit
        if (c == 'x')
            exit(0);

        printf("Enter Two Values:\n ");
        scanf("%f %f", &x, &y);
        switch (c) {
        // For Addition
        case '+':
            printf("%.1f + %.1f = %.1f\n", x, y, x + y);
            break;
        // For Subtraction
        case '-':
            printf("%.1f - %.1f = %.1f\n", x, y, x - y);
            break;
```

```
          default:
               printf(
                    "Error! please write a valid operator\n");
          }
     }
}
```

Continue:

The continue statement in C is a jump statement that is used to bring the program control to the start of the loop. We can use the continue statement in the while loop, for loop, or do..while loop to alter the normal flow of the program execution. Unlike break, it cannot be used with a C switch case.

The **C continue statement** resets program control to the **beginning** of the loop when encountered. As a result, the current iteration of the loop gets skipped and the control moves on to the next iteration. Statements after the continue statement in the loop are not executed.

Syntax of continue in C

The syntax of continue is just the continue keyword placed wherever we want in the loop body.

continue;

Use of continue in C

The continue statement in C can be used in any kind of loop to skip the current iteration. In C, we can use it in the following types of loops:

- Single Loops
- Nested Loops

Single Loops:

The continue statement can be used in for loop, while loop, and do-while loop.

```
// C program to explain the use
// of continue statement with for loop

#include <stdio.h>
int main()
{
     // for loop to print 1 to 8
     for (int i = 1; i <= 8; i++)
          {
          // when i = 4, the iteration will be skipped and for
          // will not be printed
          if (i == 4)
          {
               continue;
          }
          printf("%d ", i);
     }
     return 0;
}
```

Output

```
1 2 3 5 6 7 8
```

Nested Loops

The continue statement will only work in a single loop at a time. So in the case of nested loops, we can use the continue statement to skip the current iteration of the inner loop when using nested loops.

```
// C program to explain the use of continue statement with
nested loops
#include <stdio.h>
int main()
{
     // outer loop with 3 iterations
     for (int i = 1; i <= 3; i++)
     {
          // inner loop to print integer 1 to 4
          for (int j = 0; j <= 4; j++)
          {
               // continue to skip printing number 3
               if (j == 3)
               {
                    continue;
               }
               printf("%d ", j);
          }
          printf("\n");
     }
     return 0;
}
```

Output

```
0 1 2 4

0 1 2 4

0 1 2 4
```

goto statement

The goto statement is a jump statement which is sometimes also referred to as an **unconditional jump** statement. The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax:

```
Syntax1      |   Syntax2
----------------------------
goto label;  |    label:
.            |    .
.            |    .
.            |    .
label:       |    goto label;
```

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here, the label is a user-defined identifier that indicates the target statement. The statement immediately followed after 'label:' is the destination statement. The 'label:' can also appear before the 'goto label;' statement in the above syntax.

Program to calculate the sum and average of positive numbers

```
// If the user enters a negative number, the sum and average
are displayed.

#include <stdio.h>
#include<conio.h>
int main()
{
   const int maxInput = 100;
   int i;
   double number, average, sum = 0.0;

   for (i = 1; i <= maxInput; ++i)
     {
          printf("%d. Enter a number: ", i);
          scanf("%lf", &number);

      // go to jump if the user enters a negative number
      if (number < 0.0)
          {
               goto jump;
          }
      sum += number;
   }
jump:
   average = sum / (i - 1);
   printf("Sum = %.2f\n", sum);
   printf("Average = %.2f", average);
   return 0;
}
```

Output:

```
1. Enter a number: 3
2. Enter a number: 4.3
3. Enter a number: 9.3
4. Enter a number: -2.9
Sum = 16.60
Average = 5.53
```

Program Section:

Develop a program to accept an integer number and print whether it is palindrome or not

```
#include<stdio.h>
#include<conio.h>
int main()
{
  int n, reversed = 0, remainder, original;
    printf("Enter an integer: ");
    scanf("%d", &n);
    original = n;
    // reversed integer is stored in reversed variable
    while (n != 0)
     {
        remainder = n % 10;
        reversed = reversed * 10 + remainder;
        n /= 10;
     }
    // palindrome if orignal and reversed are equal
    if (original = = reversed)
        printf("%d is a palindrome.", original);
    else
        printf("%d is not a palindrome.", original);
   getch();
   return 0;
}
```

Output

```
Enter an integer: 1001
1001 is a palindrome.
```

Design a program to print a message 10 times.

```
// C Program to Print Hello World 10 times using For Loop
#include <stdio.h>
#include<conio.h>
int main()
{
     int i;
     for(i=0; i<10; i++)
     {
          printf("Hello World\n");
     }
return 0;
}
```

Output

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Implement a program to demonstrate logical AND operator.

```
//Program to use the Logical AND (&&) operator to check
whether the user is teenager or not.
#include <stdio.h>
#include <conio.h>
int main ()
{
    // declare variable
    int age;
    printf (" Enter the age: ");
    scanf (" %d", &age); // get age
    // use logical AND operator to check more than one
condition
    if ( age >= 13 && age <= 19)
    {
        printf (" %d is a teenager age. ", age);
    }
    else
    {
        printf (" %d is not a teenager age. ", age);
    }
    return 0;
}
```

Output

```
Enter the age: 17
17 is a teenager age.

2nd execution:
Enter the age: 10
10 is not a teenager age.
```

Write a program to print even numbers from 1 to 100

```
#include <stdio.h>
#include<conio.h>
int main(void)
{
   int n;
   printf ("Enter a number \n");
   scanf ("%d", &n);
   printf ("Even numbers from 1 to %d is : ",n);
   for (int i = 1; i <= n; i++)
     {

          if ( i % 2 == 0)
          {
             printf (" %d ", i);
          }
     }

 return 0;
}
```

Write a program to accept the value of year as input from the keyboard & print whether it is a leap year or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);
    if(((year%4==0) && ((year%400==0) || (year%100!==0))
    {
        printf("%d is a leap year", &year);
    }
     else
     {
        printf("%d is not a leap year", &year);
     }
    getch();
}
```

Output
```
Enter a year: 2004
2004 is a leap year
```

Write a program checking weather given number is prime or not.
```
#include<stdio.h>
#include<conio.h>
int main()
{
int n,i,m=0,flag=0;
printf("Enter the number to check prime:");
scanf("%d",&n);
m=n/2;
for(i=2;i<=m;i++)
{
     if(n%i==0)
     {
          printf("Number is not prime");
          flag=1;
          break;
     }
}
if(flag==0)
     printf("Number is prime");

return 0;
}
```
Output:

```
Enter the number to check prime:56
Number is not prime

Enter the number to check prime:23
Number is prime
```

Write a program to sum all the odd numbers between 1 to 20.
```
#include<stdio.h>
#include<conio.h>
void main()
{
int sum=0,i;
clrscr( );
for(i=1;i<=20;i++)
{
     if(i%2==1)
     sum=sum+i;
}
printf("sum of odd no‟s between 1 to 20 is %d”,sum);
getcht( );
}
```

Write a program for finding largest number among three numbers.

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int n1, n2, n3;
  printf("Enter three numbers: ");
  scanf("%d %d %d", &n1, &n2, &n3);
  if (n1 >= n2)
     {
          if (n1 >= n3)
              printf("%d is the largest number.", n1);
           else
              printf("%d is the largest number.", n3);
     }
  else
     {
          if (n2 >= n3)
          printf("%d is the largest number.", n2);
          else
          printf("%d is the largest number.", n3);
     }
  return 0;
}
```

Write a program to take input as a number and reverse it by while loop.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int n, reverse=0, rem;
printf("Enter a number: ");
  scanf("%d", &n);
  while(n!=0)
  {
     rem=n%10;
     reverse=reverse*10+rem;
     n/=10;
  }
  printf("Reversed Number: %d",reverse);
return 0;
}
```

Write a program to add, subtract, multiply and divide two numbers, accepted from user using switch case.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a,b;
    int op;
    printf(" 1.Addition\n 2.Subtraction\n 3.Multiplication\n
4.Division\n");
    printf("Enter the values of a & b: ");
    scanf("%d %d",&a,&b);
    printf("Enter your Choice : ");
    scanf("%d",&op);
    switch(op)
    {
    case 1:
        printf("Sum of %d and %d is : %d",a,b,a+b);
        break;
    case 2:
        printf("Difference of %d and %d is : %d",a,b,a-b);
        break;
    case 3:
        printf("Multiplication of %d and %d is : %d",a,b,a*b);
        break;
    case 4:
        printf("Division of Two Numbers is %d : ",a/b);
        break;
    default:
        printf(" Enter Your Correct Choice.");
        break;
    }
    return 0;
}
```

Write a program to calculate sum of all the odd numbers between 1 to 20.

```
// C Program To Find Sum of Odd Numbers
#include <stdio.h>
#include<conio.h>
int main()
{
int num, sum = 0;
// Asking for Input
printf("Enter the maximum value: ");
scanf("%d", &num);
printf("Odd Numbers Between 0 To %d are: \n", num);
for (int i = 1; i <= num; i++ )
{
     if (i % 2 != 0)
     {
          printf("%d\n", i);
          sum = sum + i;
     }
}
printf("The Sum of Odd Numbers From 0 To %d is %d.", num,
sum);
return 0;
}
```