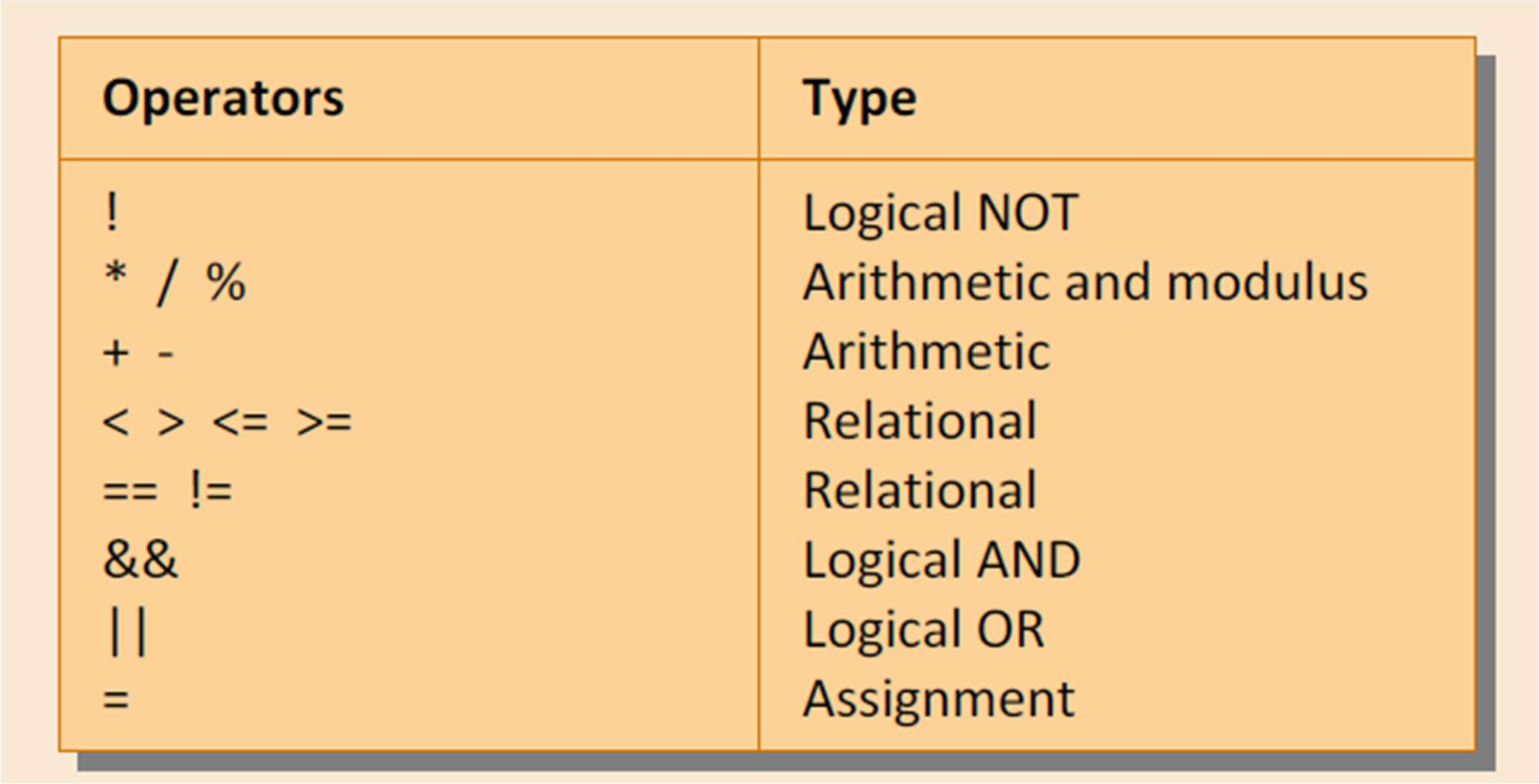
PROGRAMMING IN C

Ternary operator/Conditional operator

Operators



Operators	Type
!	Logical NOT
* / %	Arithmetic and modulus
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical AND
||	Logical OR
=	Assignment

The higher the position of an operator is in table, higher is its priority.

Logical Operator

- Another Logical operator is the NOT operator, written as **!**
- This operator reverses the result of the expression it operates on.
- If the expression evaluates to a non-zero value, then applying **!** operator to it results into a 0. And Vice Versa.
- **! (y < 10)**
- This means ‘not **y** less than 10’.
- In other words, if **y** is less than 10, the expression will be false, since **(y < 10)** is true.

Guess Output

```
# include <stdio.h>
int main( )
{
        int i ;
        printf ( "Enter value of i " ) ;
        scanf ( "%d", &i ) ;
        if ( i = 5 )
          printf ( "You entered 5\n" ) ;
        else
          printf ( "You entered something other than 5\n" ) ;
        return 0 ;
}
```

Output????

- If you entered 200

- Enter value of i 200
- You entered 5

- If you entered 9999

- Enter value of i 9999
- You entered 5

Conditional Operator

The **ternary operator** in C is a compact and efficient way to evaluate conditions and select one of two possible values. It is also referred to as the **conditional operator** and is symbolized by **? :**

(condition) ? (expression1_if_true) : (expression2_if_false);

What this expression says is: “if **condition** is true (that is, if its value is non-zero),
then the value returned will be **expression 1**,
otherwise the value returned will be **expression 2**”.

Conditional Operator

(condition) ? (expression_if_true) : (expression_if_false);

- **condition**: This is a logical or relational expression that evaluates to either true (non-zero) or false (zero).
- **expression_if_true**: This is the result or action if the condition is true.
- **expression_if_false**: This is the result or action if the condition is false.
- The operator evaluates the condition and returns either expression_if_true or expression_if_false based on the result of the condition
- The ternary operator **replaces** the need for an **if-else** statement in situations where there are two possible outcomes for a condition.

Conditional Operator

(condition) ? (expression_if_true) : (expression_if_false);

```
#include <stdio.h>
int main()
{
    int a = 10, b = 20, max;
    max = (a > b) ? a : b; // If a > b is true, max = a; otherwise, max = b.
    printf("The larger number is: %d\n", max);
    return 0;
}
```

Condition: (a > b)

If a > b is true, a is assigned to max.

If a > b is false, b is assigned to max.

Conditional Operator

```
max = (a > b) ? a : b;
```

is equivalent to:

```
        if (a > b)
                {
                        max = a;
                }
        else
                {
                        max = b;
                }
```

Conditional Operator

- It’s not necessary that the conditional operators should be used only in arithmetic statements.

Ex.:

```
 int i ;
scanf ( "%d", &i ) ;
( i == 1 ? printf ( "Amit" ) : printf ( "All and sundry" ) ) ;
```

Ex.:

```
 char a = 'z' ;
printf ( "%c", ( a >= 'a' ? a : '!' ) ) ;
```

Conditional Operator

You can use nested ternary operators to evaluate multiple conditions.

Example: Find the largest of three numbers

```
#include <stdio.h>
int main()
{
   int x = 10, y = 20, z = 15, largest;

   largest = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z);

   printf("The largest number is: %d\n", largest);
   return 0;
}
```

Limitations of the Ternary Operator

Complexity: Nested ternary operators can make the code difficult to read and debug.

Limited Use: Suitable only for two outcomes. Complex logic is better handled with if-else or switch statements.

No Block Execution: The ternary operator only allows a single statement or expression, unlike if-else, which can execute multiple statements.

Advantages of the Ternary Operator

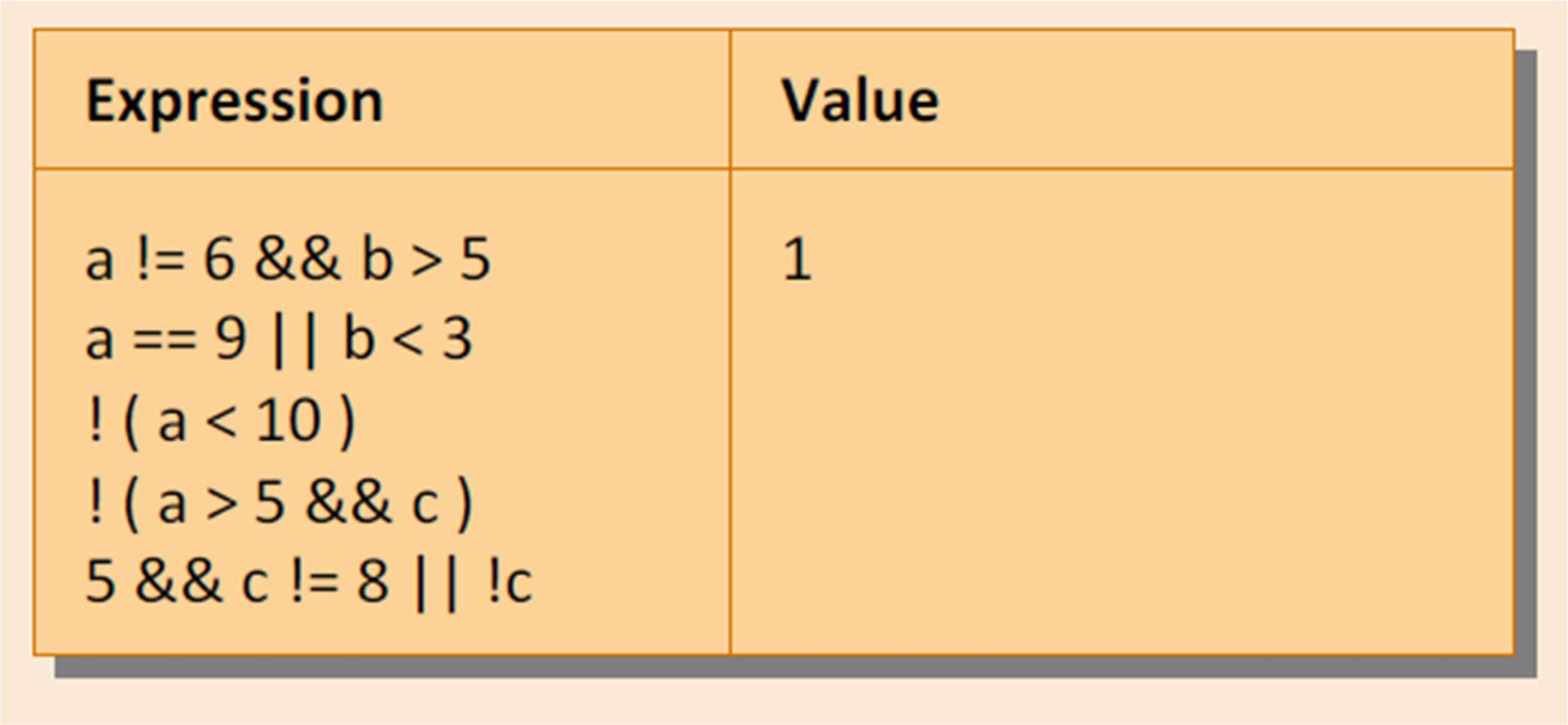
- **Conciseness**: Reduces the number of lines of code compared to if-else statements.
- **Readability**: Easy to read and understand for simple conditions.
- **Efficiency**: Suitable for straightforward decision-making processes.

Return a Grade Based on Marks

```
#include <stdio.h>
 int main()
{
 int marks = 85;
 char grade;
          grade = (marks >= 90) ? 'A' :
                        (marks >= 80) ? 'B' :
                        (marks >= 70) ? 'C' :
                        (marks >= 60) ? 'D' :
                        (marks >= 50) ? 'E' :  'F';
          printf("The grade is: %c\n", grade);
 return 0;
 }
```

Re-Cap

If a = 10, b = 12, c = 0, find the values of the expressions in the following table:



Expression	Value
a != 6 && b > 5	1
a == 9 || b < 3	
! (a < 10)	
! (a > 5 && c)	
5 && c != 8 || !c	

Re-Cap

What will be the output of the following program

```
# include <stdio.h>
int main( )
{
    int i = 4, z = 12 ;
    if ( i = 5 || z > 50 )
        printf ( "Dean of students affairs\n" ) ;
    else
        printf ( "Dosa\n" ) ;
    return 0 ;
}
```

What will be output?

```
# include <stdio.h>
int main( )
{
        int i = 1 ;
        while ( i <= 10 ) ;
        {
                printf ( "%d\n", i ) ;
                i = i + 1 ;
        }
        return 0 ;
}
```

- *Output:*
- No Output Indefinite while loop because of a ‘;’ at the end of **while**.

```
# include <stdio.h>
int main( )
{
        int i = 1 ;
        while ( i <= 10 )
        {
                printf ( "%d\n", i ) ;
                i = i + 1 ;
        }
return 0 ;
}
```

```
# include <stdio.h>
int main( )
{
        int i = 1 ;
        while ( i <= 10 )
        {
                printf ( "%d\n", i ) ;
                i ++;
        }
return 0 ;
}
```

```
# include <stdio.h>
int main( )
{
        int i = 1 ;
        while ( i <= 10 )
        {
                printf ( "%d\n", i ) ;
                i += 1;
        }
return 0 ;
}
```

```
# include <stdio.h>
int main( )
{
        int i = 0 ;
        while ( i++ < 10 )
                printf ( "%d\n", i ) ;
        return 0 ;
}
```

```
# include <stdio.h>
int main( )
{
int i = 1 ;
while ( i <= 10 ) ;
{
          printf ( "%d\n", i ) ;
           i++ ;
}
return 0 ;
}
```

Output:

No Output Indefinite while loop because of a ‘;’ at the end of **while**.

```
# include <stdio.h>
int main( )
{
        int x = 4, y, z ;
        y = --x ;
        z = x-- ;
        printf ( "%d %d %d\n", x, y, z ) ;
return 0 ;
}
```

Output:

```
2 3 3
```

Explain difference between y = ++x and y = x++

- ++x is prefix. x++ is postfix.
- In the case of y =++x , y will become (x+1), x will become (x+1). Which means that prefix adds the value before processing the data/command.
- However, in the case of y = x++ , y will become x and x will become (x+1). Which also means that postfix process the data/command first before it adds the value.

```c
# include <stdio.h>
int main( )
{
int x = 4, y = 3, z ;
z = x-- - y ;
printf ( "%d %d %d\n", x, y, z ) ;
return 0 ;
}
```

Output:

```
3 3 1
```

```
# include <stdio.h>
int main( )
{
while ( 'a' < 'b' )
printf ( " malayalam is a palindrome\n" ) ;
return 0 ;
}
```

Output:

‘malayalam is a palindrome’ will be printed indefinitely.

```
# include <stdio.h>
int main( )
{
int i ;
while ( i = 10 )
        {
                printf ( "%d\n", i ) ;
                i = i + 1 ;
        }
return 0 ;
}
```

- *Output:*
- **10 will be printed indefinitely.**

```
# include <stdio.h>
int main( )
{
float x = 1.1 ;
while ( x == 1.1 )
  {
          printf ( "%f\n", x ) ;
          x = x – 0.1 ;
  }
return 0 ;
}
```

- *Output:*
- **10 will be printed indefinitely.**

1) Write a program using conditional operators to determine whether a year entered through the keyboard is a leap year or not.
2) Write a program to find the factorial value of any number entered through the keyboard.
3) Write a program to print all the ASCII values and their equivalent characters using a while loop. The ASCII values vary from 0 to 255.
4) Write a program to enter numbers till the user wants. At the end it should display the count of positive, negative and zeros entered.

1) Write a program using conditional operators to determine whether a year entered through the keyboard is a leap year or not.

- To check if a year is a leap year, follow these rules:**Divide the year by 4**. If it is evenly divisible, it is a leap year.
- If the year is a **century year (like 1900 or 2000)**, it must also be **divisible by 400**.

```
/* Determine whether a year is leap or not */
# include <stdio.h>
int main( )
{
int year ;
printf ( "Enter Year: " ) ;
scanf ( "%d", &year ) ;
year % 100 == 0 ? ( year % 400 == 0 ? printf ( " Leap Year\n" )
: printf ( "Not a Leap Year\n" ) )
: ( year % 4 == 0 ?
printf ( "Leap Year\n" ) : printf ( "Not A Leap Year\n" ) ) ;
return 0 ;
}
```

- Write a program to print all the ASCII values and their equivalent characters using a while loop. The ASCII values vary from 32 to 255.

```
/* Print ASCII values and their corresponding characters */
# include <stdio.h>
int main( )
{
int i = 0 ;
while ( i >32 && i <= 255 )
  {
          printf ( "%d %c\n", i, i ) ;
          i++ ;
  }
return 0 ;
}
```

1) Write a program to enter numbers till the user wants. At the end it should display the count of positive, negative and zeros entered.

```
/* Count number of positives, negatives and zeros */
# include <stdio.h>
int main( )
{
int pos, neg, zero, num ;
char ans = 'y' ;
pos = neg = zero = 0 ;
while ( ans == 'y' || ans == 'Y' )
{
        printf ( "\nEnter a number: " ) ;
        scanf ( "%d", &num ) ;
        if ( num == 0 )
                zero++ ;
        if ( num > 0 )
                pos++ ;
        if ( num < 0 )
                neg++ ;
        fflush ( stdin ) ; // clears standard input stream
        printf ( "\nDo you want to continue? " ) ;
        scanf ( "%c", &ans ) ;
}
printf ( "You entered %d positive number(s)\n", pos ) ;
printf ( "You entered %d negative number(s)\n", neg ) ;
printf ( "You entered %d zero(s)\n", zero ) ;
return 0 ;
}
```