Unit # 2

# USER INTERACTION

- Use the assignment operator
- Define relational operators
- Use the following relational operators:
  - Less than (<)
  - Greater than (>)
  - Less than or equal to (<=)
  - Greater than or equal to (>=)
  - Equal to (==)
  - Not Equal to (!=)
- Define a logical operator
- Use the following logical operators:
  - AND (&&)
  - OR (||)
  - NOT (!)
- Differentiate between the assignment operator (=) and equal to operator (==)
- Differentiate between the unary and binary operators
- Define and explain the order of precedence of operators

## Unit Introduction:

A computer is a device that takes data as input, processes that data and generates the output. Thus all the programming languages must provide instructions to handle input, output and processing of data. In this chapter, we discuss different pre-built input/output functions available in C language. We also discuss different operators that we can apply to process the data.

## 2.1 Input/output (I/O) functions:

We need a way to provide input and show output while writing programs. Each programming language has its keywords or standard library functions for I/O operations. C language offers *printf* function to display the output, and *scanf* function to get input from user. In the following section, we discuss these two functions.

### 2.1.1 printf()

*printf* is a built-in function in C programming language to show output on screen. Its name comes from "print formatted" that is used to print the formatted output on screen. All data types discussed in the previous chapter can be displayed with *printf* function. To understand the working of *printf* function, let's look at the following example program:

**EXAMPLE CODE 2.1**

```
#include<stdio.h>
void main ()
{
        printf("Hello World");
}
Output:
Hello World
```

In this example, *printf* function is used to display *Hello World* on screen. Whatever we write inside the double quotes " and " in the *printf( )* function, gets displayed on screen.

**ACTIVITY 2.1**

Write down the output of following code:

```
# include<stdio.h>
void main ()
{
        printf("I am UPPERCASE and this is lowercase");
}
```

**ACTIVITY 2.2**

Write a program that shows your first name in Uppercase and your last name in lower case letters on screen.

## 2.1.2 Format Specifiers

What if we want to display the value of a variable? Let's declare a variable and then check the behavior of printf.

```
int age = 35;
```

Now I want to display the value of this variable *age* on screen. So, I write the following statement:

```
printf("age");
```

But, it does not serve the purpose, because it displays the following on screen.

```
age
```

It does not display the value stored inside the variable age, instead it just displays whatever was written inside the double quotes of *printf*. In fact, we need to specify the format of data that we want to display, using format specifiers. Table 2.1 shows format specifiers against different data types in C language.

| Data Type | Format Specifier |
|---|---|
| int | % d or % i |
| float | % f |
| char | % c |

Table 2.1: Format specifiers for I/O operations

Suppose we want to show *int* type data, we must specify it inside the *printf* by using the format specifier *%d* or *%i*. In the same way, for *float* type data we must use *%f*. It is illustrated in the following example.

**EXAMPLE CODE 2.2**

```
#include<stdio.h>
void main()
{
      float height = 5.8;
      int age = 35;
      printf("My age is %d and my height is %f", age, height);
}
```

**Output:**

```
My age is 35 and my height is 5.800000
```

We can observe that while displaying output, first format specifier is replaced with the value of first variable/data after the ending quotation mark i.e. *age* in the above example, and second format specifier is replaced with the second variable/data.

**IMPORTANT TIP**

When we use *%f* to display a float value, it displays 6 digits after the decimal point. If we want to specify the number of digits after decimal point then we can write *%.nf* where *n* is the number of digits. In the above example, if we write the following statement:

```
      printf("My age is %d and my height is %.2f", age, height);
```

The output is

```
      My age is 35 and my height is 5.80
```

**Important Note:**

Format specifiers are not only used for variables. Actually they are used to display the result of any expression involving variables, constants, or both, as illustrated in the following example.

**EXAMPLE CODE 2.3**

```
# include <stdio.h>
void main ()
{
        printf("Sum of 23 and 45 is  %d", 23 + 45);
}
```

**Output**

```
Sum of 23 and 45 is 68
```

## 2.1.3 scanf()

*scanf* is a built-in function in C language that takes input from user into the variables. We specify the expected input data type in *scanf* function with the help of format specifier. If user enters integer data type, format specifier mentioned in *scanf* must be *%d* or *%i*. Consider the following example:

**EXAMPLE CODE 2.4**

```
#include <stdio.h>
void main ()
{
        char grade;
        scanf("%c", &grade);
}
```

In this example, *%c* format specifier is used to specify character type for the input variable. Input entered by user is saved in variable *grade*.

There are two main parts of *scanf* function as it can be seen from the above code. First part inside the double quotes is the list of format specifiers and second part is the list of variables with & sign at their left.

## EXAMPLE CODE 2.5

**Example**

```
# include <stdio.h>
void main ()
{
        int number;
        printf("Enter a number between 0-10: ");
        scanf("%d", &number);
        printf("The number you entered is: %d", number);
}
```

**Output:**

```
Enter a number between 0-10: 4
The number you entered is: 4
```

### Important Note

We can take multiple inputs using a single scanf function e.g. look at the following statement.

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

It takes input into two integer type variables *a* and *b*, and one float type variable *c*. After each input, user should enter a *space* or press *enter* key. After all the inputs user must press *enter* key.

## ACTIVITY 2.3

Write a program that takes roll number, percentage of marks and grade from user as input. Program should display the formatted output like following:

```
Roll No     :          input value
Percentage  :          input value %
Grade       :          input value
```

### Important Note

It is a very common mistake to forget & sign in the *scanf* function. Without & sign, the program gets executed but does not behave as expected.

### 2.1.4 getch()

*getch()* function is used to read a character from user. The character entered by user does not get displayed on screen. This function is generally used to hold the execution of program because the program does not continue further until the user types a key. To use this function, we need to include the library *conio.h* in the header section of program.

**EXAMPLE CODE 2.6**

```
# include<stdio.h>
# include<conio.h>
void main ()
{
        printf("Enter any key if you want to exit program ");
        getch();
}
```

In the above, program prompts user to enter a character and then waits for the user's input before finishing the execution of program.

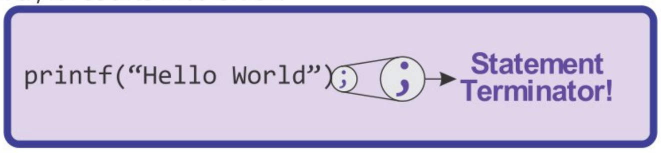
**EXAMPLE CODE 2.7**

```
# include<stdio.h>
# include<conio.h>
void main ()
{
        char key;
        printf("Enter any key : ");
        key = getch(); //Gets a character from user into variable key
}
```

If we run this program, we notice a difference between reading a character using *scanf* and reading a character using *getch* functions. When we read character through *scanf*, it requires us to press enter for further execution. But in case of *getch*, it does not wait for enter key to be pressed. Function reads a character and proceeds to the execution of next line.

## 2.1.5 Statement Terminator

A statement terminator is identifier for compiler which identifies end of a line. In C language *semi colon (;)* is used as statement terminator. If we do not end each statement with a ; it results into error.



```
printf("Hello World");
```

## 2.1.6 Escape Sequence

### Purpose

Escape sequences are used in *printf* function inside the " and ". They force *printf* to change its normal behavior of showing output. Let's understand the concept of an escape sequence by looking at the following example statement:

```
printf("My name is \"Ali\"");
```

The output of above statement is

```
My name is "Ali"
```

In the above example \" is an escape sequence. It causes *printf* to display " on computer screen.

### Formation of escape sequence

Escape sequences consist of two characters. The first character is always back slash (\) and the second character varies according to the functionality that we want to achieve. Back slash (\) is called escape character which is associated with each escape sequence to notify about escape. Escape character and character next to it are not displayed on screen, but they perform specific task assigned to them.

**DID YOU KNOW?**

Besides different escape sequences discussed in this section, following escape sequences are also commonly used in C language.

| Sequence | Purpose | Sequence | Purpose |
|---|---|---|---|
| **\\'** | Displays Single Quote( ' ) | **\a** | Generates an alert sound |
| **\\** | Displays Back slash( \ ) | **\b** | Removes previous char |

## New Line (\n)

After escape character, *n* specifies movement of the cursor to start of the next line. This escape sequence is used to print the output on multiple lines. Consider the following example to further understand this escape sequence:

**EXAMPLE CODE 2.8**

```
#include <stdio.h>
void main ()
{
      printf("My name is Ali. \n");
      printf("I live in Lahore.");
}
```

**Output**

```
My name is Ali.
I live in Lahore.
```

**Important Note:**

In the absence of an escape sequence, even if we have multiple *printf* statements, their output is displayed on a single line. Following example illustrates this point..

**EXAMPLE CODE 2.9**

```
#include<stdio.h>
void main()
{
      printf("My name is");
      printf(" Ahmad");
}
```

**Output**

```
My name is Ahmad
```

## Tab (\t)

Escape sequence \t specifies the I/O function of moving to the next tab stop horizontally. A tab stop is collection of 8 spaces. Using \t takes cursor to the next tab stop. This escape sequence is used when user presents data with more spaces

**</> EXAMPLE CODE 2.10**

```
#include<stdio.h>
void main ()
{
      printf("Name: \tAli\nFname: \tHammad\nMarks: \t1000");
}
```

**Output**

```
Name:   Ali
Fname:  Hammad
Marks:  1000
```

## 2.2 Operators

The name *computer* suggests that computation is the most important aspect of computers. We need to perform computations on data through programming. We have a lot of mathematical functions to perform calculations on data. We can also perform mathematical operations in our programs. C language offers numerous operators to manipulate and process data. Following is the list of some basic operator types:

- Assignment operator
- Arithmetic operators
- Logical operators
- Relational operators

### 2.2.1 Assignment Operator

Assignment operator is used to assign a value to a variable, or assign a value of variable to another variable.

*Equal sign (=)* is used as assignment operator in C. Consider the following example:

```
int sum = 5;
```

Value 5 is assigned to a variable named *sum* after executing this line of code.

Let's have a look at another example:

```
int sum = 6;
int var = sum;
```

First, value 6 is assigned to variable *sum*. In the next line, the value of *sum* is assigned to variable *var*.

**PROGRAMMING TIME 2.1**

Write a program that swaps the values of two integer variables.

**Program:**

```
void main()
{
        int a = 2, b = 3, temp;
        temp = a;
        a = b;
        b = temp;
        printf("Value of a after swapping: %d\n", a);
        printf("Value of b after swapping: %d\n", b);
}
```

## 2.2.2 Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on data. Table 2.2 represents arithmetic operators with their description.

| Operator | Name | Description |
|---|---|---|
| / | Division Operator | It is used to divide the value on left side by the value on right side. |
| * | Multiplication Operator | It is used to multiply two values. |
| + | Addition Operator | It is used to add two values. |
| - | Subtraction Operator | It is used to subtract the value on right side from the value on left side. |
| % | Modulus Operator | It gives remainder value after dividing the left operand by right operand. |

Table 2.2: Arithmetic Operators

### Division

Division operator (/) divides the value of left operand by the value of right operand. e.g. look at the following statement.

```
    float result = 3.0 / 2.0;
```

After this statement, the variable *result* contains the value 1.5.

**Important Note**

If both the operands are of type *int*, then result of division is also of type *int*. Remainder is truncated to give the integer answer. Consider the following line of code:

```
        float result =  3 / 2;
```

As both values are of type *int* so answer is also an integer which is 1. When this value 1 is assigned to the variable *result* of type *float*, then this 1 is converted to *float*, so value 1.0 is stored in variable *result*. If we want to get the precise answer then one of the operands must be of floating type. Consider the following line of code:

```
        float result = 3.0 / 2;
```

In the above example, the value stored in variable *result* is 1.5.

**PROGRAMMING TIME 2.2**

```
/*This program takes as input the price of a box of chocolates and
the total number of chocolates in the box. The program finds and
displays the price of one chocolate.*/
# include <stdio.h>
void main ()
{
    float box_price, num_of_chocolates, unit_price;
    printf (“Please enter the price of whole box of chocolates: ”);
    scanf(“%f”, &box_price);
    printf (“Please enter the number of chocolates in the box: ”);
    scanf(“%f”, &num_of_chocolates);
    unit_price = box_price / num_of_chocolates;
    printf(“The price of a single chocolate is %f”, unit_price);
}
```

**Output:**

```
Please enter the price of whole box of chocolates: 150
Please enter the number of chocolates in the box: 50
The price of a single chocolate is 3.000000
```

## Multiplication

Multiplication operator (*) is a binary operator which performs the product of two numbers. Look at the following statement:

```
int multiply = 5 * 5;
```

After the execution of statement, the variable *multiply* contains value 25.

### PROGRAMMING TIME 2.3

```
/* Following program takes as input the length and width of a
rectangle. Program calculates and displays the area of rectangle on
screen. */
# include<stdio.h>
void main ()
{
      float length, width, area;
      printf("Please enter the length of rectangle: ");
      scanf("%f", &length);
      printf("Please enter the width of rectangle: ");
      scanf("%f", &width);
      area = length * width;
      printf("Area of rectangle is : %f", area);
}
```

**Output**

```
Please enter the length of rectangle: 6.5
Please enter the length of rectangle: 3
Area of rectangle is : 19.500000
```

### ACTIVITY 2.4

Write a program that takes as input the length of one side of a square and calculates the area of square.

## Addition

Addition operator (+) calculates the sum of two operands. Let's look at the following statement:

```
int add = 10 + 10;
```

Resultant value in variable *add* is 20.

**PROGRAMMING TIME 2.4**

```
/* This program takes marks of two subjects from user and displays
the sum of marks on console. */
#include <stdio.h>
void main ()
{
      int sum, math, science;
      printf("Enter marks of Mathematics: ");
      scanf("%d", &math);
      printf("Enter marks of Science: ");
      scanf("%d", &science);
      sum = math + science;
      printf("Sum of marks is : %d", sum);
}
```

**Output**

```
Enter marks of Mathematics:  90
Enter marks of Science: 80
Sum of marks is : 170
```

**ACTIVITY 2.5**

Write a program that takes as input the number of balls in jar A and the number of balls in jar B. The program calculates and displays the total number of balls.

**IMPORTANT TIP**

The statement `a = a + 1;` is used to increase the value of variable *a* by 1. In C language, this statement can also be written as `a++;` or `++a;`. Similarly, `a--;` or `--a;` is used to decrease the value of *a* by 1.

### Subtraction

Subtraction operator (–) subtracts right operand from the left operand. Let's look at the following statement:

```
int result = 20 – 15;
```

After performing subtraction, value 5 is assigned to the variable *result*.

**ACTIVITY 2.6**

Write a program that takes original price of a shirt and discount percentage from user. Program should display the original price of shirt, discount on price and price after discount.

### Modulus operator

Modulus operator (%) performs division of left operand by the right operand and returns the remainder value after division. Modulus operator works on integer data types.

```
int remaining = 14 % 3;
```

As, when we divide 14 by 3, we get a remainder of 2, so the value stored in variable *remaining* is 2.

**PROGRAMMING TIME 2.5**

```
/* This program finds and displays the right most digit of an input
number. */
#include <stdio.h>
void main()
{
        int num, digit;
        printf("Enter a number: ");
        scanf("%d", &num);
        digit = num % 10;
        printf("Right most digit of number you entered is: %d",
        digit);
}
```

**Output**

```
Enter a number: 789
Right most digit of number you entered is : 9
```

**ACTIVITY 2.7**

Write a program that takes 2 digit number from user, computes the product of both digits and show the output.

**ACTIVITY 2.8**

Write a program that takes seconds as input and calculates equivalent number of hours, minutes and seconds.

**Important Note:**

While writing arithmetic statements in C language, a common mistake is to follow the usual algebraic rules e.g. writing 6 * y as 6y, and writing x * x * x as $x^3$ etc. It results in a compiler error.

**ACTIVITY 2.9**

Convert the following algebraic expressions into C expressions.

$x = 6y + z$

$x = yz^3 + 3y$

$z = x + \frac{y^2}{3x}$

$z = (x - 2)^2 + 3y$

$y = (x + \frac{3z}{2}) + z^3 + \frac{x}{z}$

## 2.2.3 Relational Operators

Relational operators compare two values to determine the relationship between values. Relational operators identify either the values are equal, not equal, greater than or less than one another. C language allows us to perform relational operators on numeric and char type data. Table 2.3 presents relational operators in C language and their descriptions:

| Relational Operator | Description |
|---|---|
| = = | Equal to |
| ! = | Not equal |
| > | Greater than |
| < | Less than |
| > = | Greater than equal to |
| < = | Less than equal to |

Table 2.3: Basic relational operators with their description

Relational operators perform operations on two operands and return the result in Boolean expression (*true* or *false*). A *true* value is represented by 1, whereas a *false* value is represented by a 0. This concept is further illustrated in Table 2.4.

| Relational Expression | Explanation | Result |
|---|---|---|
| 5 = = 5 | 5 is equal to 5? | True |
| 5 ! = 7 | 5 is not equal to 7? | True |
| 5 > 7 | 5 is greater than 7? | False |
| 5 < 7 | 5 is less than 7? | True |
| 5 > = 5 | 5 is greater than or equal to 5? | True |
| 5 < = 4 | 5 is less than or equal to 4? | False |

Table 2.4: Illustration of relational operators with examples

### 2.2.4 Assignment operator (=) and equal to operator (= =):

In C language, == operator is used to check for equality of two expressions, whereas = operator assigns the result of expression on right side to the variable on left side. Double equal operator (==) checks whether right and left operands are equal or not. Single equal operator (=) assigns right operand to the variable on left side.

**Important Note**

We can also use printf function to show the results of a relational expression, e.g. look at the following examples:

```
printf("%d", 5 == 5); // This statement displays 1
printf("%d", 5 > 7);  // This statement displays 0
```

**ACTIVITY 2.10**

Consider the variables x=3, y=7. Find out the Boolean result of following expressions.

| | |
|---|---|
| *(2 + 5) > y* | *(x + 4) == y* |
| *x != (y - 4)* | *(y / 2) >= x* |
| *-1 < x* | *(x * 3) <= 20* |

## 2.2.5 Logical Operators

Logical operators perform operations on Boolean expressions and produce a Boolean expression as a result.

As we have studied that result of a relational operation is a Boolean expression, so logical operators can be performed to evaluate more than one relational expressions. Table 2.5 shows the logical operators offered by C language.

| Operator | Description |
|---|---|
| && | Logical AND |
| \|\| | Logical OR |
| ! | Logical NOT |

Table 2.5: Basic logical operators and their description

### AND operator (&&):

AND operator && takes two Boolean expressions as operands and produces the result *true* if both of its operands are *true*. It returns *false* if any of the operands is *false*. Table 2.6 shows the **truth table** for AND operator.

| Expression | Result |
|---|---|
| False && False | False |
| False && True | False |
| True && False | False |
| True && True | True |

Table 2.6: Truth table for AND operator

## OR operator (||):

OR operator accepts Boolean expression and returns *true* if at least one of the operands is *true*. Table 2.7 shows the truth table for OR operator.

| Expression | Result |
|---|---|
| False \|\| False | False |
| False \|\| True | True |
| True \|\| False | True |
| True \|\| True | True |

Table 2.7: Truth table for OR operator

## NOT operator (!):

NOT operator negates or reverses the value of Boolean expression. It makes it *true*, if it is *false* and *false* if it is *true*. Table 2.8 presents the truth table for Not operator.

| Expression | Result |
|---|---|
| !(True) | False |
| !(False) | True |

Table 2.8: Truth table for NOT operator

## Examples of Logical Operators:

Table 2.9 illustrates the concept of logical operators with the help of examples.

| Logical Expression | Explanation | Result |
|---|---|---|
| 3 < 4 && 7 > 8 | 3 is less than 4 AND 7 is greater than 8? | False |
| 3 = = 4 \|\| 3 > 1 | 3 is equal to 4 OR 3 is greater than 1? | True |
| ! (4 > 2 \|\| 2 = = 2) | NOT (4 is greater than 2 OR 2 is equal to 2)? | False |
| 6 < = 6 && !(1 > 2) | 6 is less than or equal to 6 AND NOT (1 is greater than 2)? | True |
| 8 > 9 \|\| !(1 < =0) | 8 is greater than 9 OR NOT (1 is less than or equal to 0)? | True |

Table 2.9: Illustration of logical operators with examples

**DID YOU KNOW?**

C language performs **short-circuit evaluation**. It means that:

**1-** While evaluating an AND operator, if sub expression at left side of the operator is *false* then the result is immediately declared as *false* without evaluating complete expression.

**2-** While evaluating an OR operator, if sub expression at left side of the operator is *true* then the result is immediately declared as true without evaluating complete expression.

**ACTIVITY 2.11**

Assume the following variable values x=4, y=7, z=8. Find out the resultant expression.

```
x== 2  ||  y==8                    7>= y   &&  z<5
z>=5   ||  x<= -3                  y==7   &&  !(true)
x!=y    ||   y<5                   !(z>x)
```

## 2.2.6 Unary vs Binary Operators:

All the operators discussed in this chapter can be divided into two basic types, based on the number of operands on which the operator can be applied.

**Unary Operators:** Unary operators are applied over one operand only e.g. logical not (!) operator has only one operand. Sign operator (-) is another example of a unary operator e.g. -5.

**Binary Operators:** Binary operators require two operands to perform the operation e.g. all the arithmetic operators, and relational operators are binary operators. The logical operators && and || are also binary operators.

**DID YOU KNOW?**

C language also offers a ternary operator that is applied on three operands.

## 2.2.7 Operators' Precedence:

If there are multiple operators in an expression, the question arises that which operator is evaluated first. To solve this issue, a precedence has been given to each operator (Table 2.10). An operator with higher precedence is evaluated before the operator with lower precedence. In case of equal precedence, the operator at left side is evaluated before the operator at right side.

**Example:**

```
result = 18  / 2 * 3 + 7 % 3 + ( 5 * 4);     // evaluate ( )
result = 18  / 2 * 3 + 7 % 3 + 20;           // evaluate /
result = 9 * 3 + 7 % 3 + 20;                 // evaluate *
result = 27 + 7 % 3 + 20;                    // evaluate %
result = 27 + 1 + 20;                        // evaluate +
result = 28 + 20;                            // evaluate +
result = 48;
```

| Operator | Precedence |
|---|---|
| ( ) | 1 |
| ! | 2 |
| * , / , % | 3 |
| + , - | 4 |
| > , < , >= , <= | 5 |
| == , != | 6 |
| & & | 7 |
| \|\| | 8 |
| = | 9 |

Table 2.10: Operators with their precedence

**ACTIVITY 2.12**

Find out the results of the following Expressions:

**16 / (5 + 3)**
**7 + 3 * (12 + 2)**
**25 % 3 * 4**
**34 – 9 * 2 / (3 * 3)**
**18 / (15 – 3 * 2)**

## SUMMARY

- We need a way to provide input and show output while writing programs. Each programming language has its keywords or standard built-in function for I/O operations.
- ***printf*** is a built-in function in C programming language. It's name comes from "print formatted" that is used to show the formatted output on screen.
- ***Format specifiers*** are used to specify format of data type during input and output operations. Format specifier is always preceded by a percentage (%) sign.
- ***scanf*** is a built-in function in C language that takes input from user into the variables
- ***getch()*** function is used to read a character from user. This function accepts characters only. The character entered by user does not get displayed on screen.
- A ***statement terminator*** is identifier for compiler which identifies end of a statement. In C language *semi colon (;)* is used as statement terminator.
- ***Escape sequence*** forces printf to escape from its normal behavior. It is the combination of escape character(\) and a character associated with special functionality.
- Escape sequence \*n* specifies the movement of cursor to start of the next line. This escape sequence is used to display the output on multiple lines.
- Escape sequence \*t* specifies the movement of cursor to the next tab stop horizontally. A tab stop is collection of 8 spaces.
- ***Basic operators*** are arithmetic operators, assignment operator, relational operators, and logical operators.
- ***Arithmetic operators*** Arithmetic operators are used to perform arithmetic operations on data to evaluate arithmetic functions. Arithmetic operators are +, -, *, /, %.

- ***Modulus operator*** is also a binary operator, which performs division of left operand to the right operand and returns the remainder value after division. Modulus operator works on integer data types.
- ***Relational operators*** compare two values to determine the relationship between values.
- ***Logical operator*** performs operation on Boolean expressions and returns a Boolean value as a result.
- ***Logical AND operator*** returns *true* when the result of expressions on both sides is *true* whereas the ***Logical OR operator*** returns *true* when either of the two expressions is true.
- ***Logical NOT operator*** returns *true* if the expression is *false* and vice versa.
- ***Short circuiting*** is to deduce the result of an operation without computing the whole expression.
- There are three types of operators. ***Unary, binary and ternary operators*** require one, two and three operands respectively, to perform the operation.
- ***Precedence*** tells which operation should be performed first. Different operators have different precedence. Operators with higher precedence are evaluated first and the ones with lowest precedence are evaluated last.

## Exercise

### Q.1 Multiple Choice Questions

**1)** printf is used to print __________ type of data.

**a)** *int* **b)** *float* **c)** *char* **d)** All of them

**2)** scanf is a _________ in C programming language.

**a)** Keyword **b)** library **c)** function **d)** none of them

**3)** getch() is used to take ________ as input from user.

**a)** *int* **b)** *float* **c)** *char* **d)** All of them

**4)** Let the following part of code, what will be the value of variable ***a*** after execution:

```
int a = 4;
float b = 2.2;
a = a * b;
```

**a)** 8.8 **b)** 8 **c)** 8.0 **d)** 8.2

**5)** Which of the following is a valid line of code.

**a)** int = 20; **b)** grade= 'A'; **c)** line= this is a line; **d)** none of them

**6)** Which operator has highest precedence among the following:

**a)** / **b)** = **c)** > **d)** !

**7)** Which of the following is not a type of operator:

**a)** Arithmetic operator **c)** Relational operator

**b)** Check operator **d)** Logical operator

**8)** The operator % is used to calculate ____________.

**a)** Percentage **b)** Remainder **c)** Factorial **d)** Square

**9)** Which of the following is a valid character:

**a)** 'here' **b)** "a" **c)** '9' **d)** None of them

**10)** What is true about C language:

**a)** C is not a case sensitive language

**b)** Keywords can be used as variable names

**c)** All logical operators are binary operators

**d)** None of them

## Q.2 True or False

**1)** Maximum value that can be stored by an integer is 32000. T/F

**2)** Format specifiers begin with a % sign. T/F

**3)** Precedence of division operator is greater than multiplication operator. T/F

**4)** *getch* is used to take all types of data input from user. T/F

**5)** *scanf* is used for output operations. T/F

## Q.3 Define the following.

**1)** Statement Terminator **2)** Format Specifier **3)** Escape Sequence
**4)** scanf **5)** Modulus Operator

## Q.4 Briefly answer the following questions.

**1)** What is the difference between *scanf* and *getch*?

**2)** Which function of C language is used to display output on screen?

**3)** Why format specifiers are important to be specified in I/O operations?

**4)** What are escape sequences? Why do we need them?

**5)** Which operators are used for arithmetic operations?

**6)** What are relational operators? Describe with an example.

**7)** What are logical operators? Describe with an example.

**8)** What is the difference between unary operators and binary operators?

**9)** What is the difference between == operator and = operator?

**10)** What is meant by precedence of operators? Which operator has the highest precedence in C language?

## Q.5 Write down output of the following code segments.

**a)**
```
# include<stdio.h>
void main ()
{
        int x = 2, y = 3, z = 6;
        int ans1, ans2, ans3;
        ans1 = x / z * y ;
        ans2 = y + z / y * 2;
        ans3 = z / x + x * y;
        printf("%d %d %d", ans1, ans2, ans3 );
}
```

**b)**
```
# include<stdio.h>
void main ()
{
        printf ( "nn \n\n nnn\nn\nt\t" ) ;
        printf ( "nn /n/n nn/n\n" ) ;
}
```
**c)**
```
#include<stdio.h>
void main()
{
        int a = 4, b;
        float c = 2.3;
        b = c * a;
        printf("%d", b);
}
```
**d)**
```
#include<stdio.h>
void main()
{
        int a = 4 * 3 / ( 5 + 1 ) + 7 % 4;
        printf("%d", a);
}
```
**e)**
```
#include<stdio.h>
void main()
{
        printf("%d", (( ( 5 > 3 ) && (4 > 6) ) || (7 > 3)) );
}
```

## Q.6 Identify errors in the following code segments.

**a)**
```
#include<stdio.h>
void main ()
{
        int a , b = 13;
        b = a % 2;
        printf("Value of b is : %d, b);
}
```

**b)**
```
#include<stdio.h>
void main ()
{
      int a , b , c,
      printf("Enter First Number: ");
      scanf("%d", &a);
      printf("Enter second number : ");
      scanf("%d", &b);
      a + b = c;
]
```
**c)**
```
#include<stdio.h>;
main ()
{
      int num;
      printf(Enter number: ");
      scanf(%d, &num);
};
```
**d)**
```
include<stdio.h>
int main ()
{
      float f;
      printf["Enter value: "];
      scanf("%c", &f);
}
```

## Programming Exercises

### Exercise 1

The criteria for calculation of wages in a company is given below.

| | | | |
|---|---|---|---|
| *Basic Salary* | *= Pay Rate Per Hour* | X | *Working Hours Of Employee* |
| *Overtime Salary* | *= Overtime Pay Rate* | X | *Overtime Hours Of Employee* |
| *Total Salary* | *= Basic Salary* | + | *Overtime Salary* |

Write a program that should take working hours and overtime hours of employee as input. The program should calculate and display the total salary of employee.

## Exercise 2

Write a program that takes Celsius temperature as input, converts the temperature into Fahrenheit and shows the output. Formula for conversion of temperature from Celsius to Fahrenheit is:

$$F = \frac{9}{5}C + 32$$

## Exercise 3

Write a program that displays the following output using single *printf* statement:

```
*        *        *        *
1        2        3        4
```

## Exercise 4

Write a program that displays the following output using single *printf* statement:

```
I am a Boy
I live in Pakistan
I am a proud Pakistani
```

## Exercise 5

A clothing brand offers 15% discount on each item. A lady buys 5 shirts from this brand. Write a program that calculates total price after discount and amount of discount availed by the lady. Original prices of the shirts are:

Shirt1 = 423
Shirt2 = 320
Shirt3 = 270
Shirt4 = 680
Shirt5 = 520

**Note:** Use 5 variables to store the prices of shirts.

## Exercise 6

Write a program that swaps the values of two integer variables without help of any third variable.

## Exercise 7

Write a program that takes a 5 digit number as input, calculates and displays the sum of first and last digit of number.

## Exercise 8

Write a program that takes monthly income and monthly expenses of the user like electricity bill, gas bill, food expense. Program should calculate the following:

- Total monthly expenses
- Total yearly expenses
- Monthly savings
- Yearly saving
- Average saving per month
- Average expense per month

## Exercise 9

Write a program that takes a character and number of steps as input from user. Program should then jump number of steps from that character.

**Sample output:**

```
Enter character: a
Enter steps: 2
New character : c
```

## Exercise 10

Write a program that takes radius of a circle as input. The program should calculate and display the area of circle.