Contents

Unit 4
Operators, Expressions, and Input/Output Functions

Structure of the Unit

4.1. Unit Outcomes

After the successful completion of this unit, the student will be able to:

1. Explain the classification of operators.
2. Describe the precedence and associativity of operators.
3. Discuss the input-output function used for data.
4. Develop programs using different data types.

4.2. Operators and Expression

The operators are used to performing different arithmetic and logical operations. To perform different operations such as arithmetic, comparison, and logical operations, built-in operators are used in the C program. The operators in C are classified as follows:

1) Arithmetic Operators
2) Relational Operators
3) Logical Operators
4) Bitwise Operators
5) Assignment Operators
6) Miscellaneous Operators

a) Arithmetic Operators: The arithmetic operators are given in Table 3.2.

Table 3.2: Arithmetic Operators

Operators	Description	Example
+	Addition	`c = a + b`

-	Subtraction	c = a - b
*	Multiplication	c = a * b
/	Division	c = a / b
%	Modulus: Returns remainder	c = a % b
++	Increment	a++
--	Decrement	b++

b) Relational Operators: The relational operators are given in Table 3.3

Table 3.2: Relational Operators

Operators	Description	Example
==	Checks if the values of two operands are equal or not, if yes then the condition becomes true.	(a==b)
!=	Checks if the values of two operands are equal or not, if values are not equal then the condition becomes true	(a!=b)
>	Checks if the value of the left operand is greater than the value of the right operand, if yes then the condition becomes true.	(a>b)
<	Checks if the value of the left operand is less than the value of the right operand, if yes then the condition becomes true.	(a<b)
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes then the condition becomes true.	(a>=b)
<=	Checks if the value of the left operand is less than or equal to the value of the right operand, if yes then the condition becomes true.	(a<=b)

1) Logical Operators: The list of logical operators is given in Table 2.3.

Table 3.3: Logical Operators

Operators	Description	Example
&&	Called Logical AND operator. If both operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two	(A || B) is true.

	operands are non-zero, then the condition becomes true.	
!	Called Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make false.	!(A && B) is true.

2) Bitwise operators: Bitwise operators are used to perform bit-by-bit operations. Bitwise operations are based on the logic gates AND, OR, and EX-OR. The operations in logic gates are given in Table 3.4.

Table 3.4: Logical Operator's Operations

a	b	a & b (AND)	a | b (OR)	a ^ b (EX-OR)
0	0	0	0	0
0	1	1	1	1
1	0	0	1	0
1	1	1	1	0

Table 3.5: Bit-wise Operators

Bitwise Operators		
For all examples below consider a = 10 and b = 5		
Operator	**Description**	**Example**
&	Bitwise AND	a & b gives 0
|	Bitwise OR	a | b gives 15
^	Bitwise Ex-OR	a ^ b gives 15
~	1's complement (NOT)	~a gives some negative value
<<	Left shift	a << 1 gives 20
>>	Right shift	a >> 1 gives 5

3) Assignment can be combined with the other operators as expressed in Table 3.6.

Table 3.6 Assignment Operator Combination

Operator	Description	Example
=	A simple assignment operator assigns values from right-side operands to left-side operands.	`C = A + B will assign the value of A + B to C`

+=	Add with the assignment operator, it adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract with the assignment operator, it subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	The multiply and assignment operator multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	The divide AND assignment operator divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	The modulus AND assignment operator takes the modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

4) Some of the miscellaneous operators in C are given in Table 3.7.

Table 3.7: Miscellaneous operators

S.No	Operator & Description
1	sizeof sizeof operator returns the size of a variable. For example, the sizeof(a), where a is an integer, will return 4.
2	Condition? X: Y

	(This is known as a Ternary or Conditional Operator) It is a conditional operator (?). If Condition is true, then it returns the value of X otherwise returns the value of Y.
3	, The comma operator causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
4	`. (dot) and -> (arrow)` Member operators are used to reference individual members of classes, structures, and unions.
5	Cast Casting operators convert one data type to another. For example, int(2.2000) would return 2.
6	**&** Pointer operator & returns the address of a variable. For example, &a; will give the actual address of the variable.
7	• The pointer operator * is a pointer to a variable. For example, *var; will pointer to a variable var.

4.2.1 Operator Precedence

When there are many operators in the expression, the precedence and associativity of the operators are used to determine the priority and are helpful in the evaluation of the expression by the compiler. With these rules, there will not be any confusion in solving the expressions. The rules are based on basic mathematics and are incorporated into the compiler. Table 3.8 lists the different operator precedence and the direction of its evaluation.

Table 3.8 Operator Precedence in C

Precedence	Operator	Meaning
1	`()`	Parenthesis (for functions)
	`[]`	Square brackets (for array)
	`.`	Dot operator
	`->`	Structure member access using pointers
	`++, --`	Post-increment and decrement
	`++/--`	Pre-increment and decrement
	`+ -`	Unary plus and minus

	! ~	Logical NOT, Bitwise Complement
2	(type)	Cast Operator
	*	Dereferencing Operator
	&	Address of Operator
	sizeof	Determining size in bytes
3	*, /, %	Multiplication, Division, and Modulus
4	+, -	Addition and Subtraction
5	<<, >>	Bitwise Shift Left, Bitwise Shift Right
6	<<, <<=	Relational Operators: Less Than, Less Than Equal to
7	>>, >>=	Greater Than, Greater Than Equal To
8	&	Bitwise AND
9	^	Bitwise Exclusive OR
10	|	Bitwise Inclusive OR
11	&&	Logical AND
12	||	Logical OR
13	?:	Ternary or Conditional Operator
14	=	Assignment
	+=, -=	Addition, Subtraction Assignment
	^=, |=	Bitwise exclusive, inclusive-OR assignment
	<<=, >>=	Bitwise shift left, right assignment
15	,	The comma (expression separator)

Examples: The evaluation of the expression is as per the precedence given below.
5+10∗3=¿35
In this expression, there are two operators, + (plus), and * (multiply). According to the given table, the * has higher precedence than + so, the first evaluation will be 10∗3 and the next will be 5+30. Some more examples of operator precedence are as follows.

4.2.2 Associativity in Expression

When two operators have the same precedence in an expression, then the associativity decides whether to evaluate the expression from left to right or vice versa. The associativity rules are shown in Table 3.9.

Table 3.9 Associativity Rules

Precedence	Operator	Meaning	Associativity
1	()	Parenthesis (for functions)	Left to Right
	[]	Square brackets (for array)	
	.	Dot operator	
	->	Structure member access using pointers	

	++, --	Post-increment and decrement	
	++/--	Pre-increment and decrement	
	+ -	Unary plus and minus	
	! ~	Logical NOT, Bitwise Complement	
2	(type)	Cast Operator	Right to Left
	*	Dereferencing Operator	
	&	Address of Operator	
	sizeof	Determining size in bytes	
3	*, /, %	Multiplication, Division, and Modulus	Left to Right
4	+, -	Addition and Subtraction	Left to Right
5	<<, >>	Bitwise Shift Left, Bitwise Shift Right	Left to Right
6	<<, <<=	Relational Operators: Less Than, Less Than Equal to	Left to Right
7	>>, >>=	Greater Than, Greater Than Equal To	Left to Right
8	&	Bitwise AND	Left to Right
9	^	Bitwise Exclusive OR	Left to Right
10	|	Bitwise Inclusive OR	Left to Right
11	&&	Logical AND	Left to Right
12	||	Logical OR	Left to Right
13	?:	Ternary or Conditional Operator	Left to Right
14	=	Assignment	Right to Left
	+=, -=	Addition, Subtraction Assignment	
	^=, |=	Bitwise exclusive, inclusive-OR assignment	
	<<=, >>=	Bitwise shift left, right assignment	
15	,	The comma (expression separator)	Left to Right

Examples: Consider an expression 20/5%2.
There are two operators / and % with the same precedence. In such a case the rules of associativity are applied. In this case, the expression is evaluated from left to right.
20/5%2

4%2

0

```
#include <stdio.h>

int main()
{
int a = 50, b = 50, c = 100;
printf("%d == %d is %d \n", a, b, a == b);
printf("%d == %d is %d \n", a, c, a == c);
return 0;
  }
Output:
50 == 50 is 1
   50 == 100 is 0
```

```
#include <stdio.h>
int main()
{
int a = 50, b = 50, c = 100;
printf("%d > %d is %d \n”, a, c,a > c);
printf("%d < %d is %d \n", b, c, b < c);
printf("%d != %d is %d \n", a, c, a != c);
printf("%d >= %d is %d \n", a, c, a >= c);
return 0;
}
Output:
50 > 100 is 0
50 < 100 is 1
50 != 100 is 1
50 >= 100 is 0
```

```
// C program for Logical  AND Operator
#include <stdio.h>
int main()
{
      int marks = 80;
      if (marks > 50 && marks<90)
      printf("First class \n");
      return 0;
 }
```

```
#include<stdio.h>
int main() {
int x = 5, y = 10;
void function_add(x, y);
}
```

4.3. Input-Output Functions

Entering data into the program and displaying or sending data from the program is referred to as an input/output (I/O) operation. These I/O operations are supported by a standard library file in C language, namely stdio. h. C files are also known as I/O streams. There are three I/O streams as given in Table 3.10.

Table 3.10. I/O Streams in C Programming

Standard File	Name	Meaning
Standard input	stdin	used to take the input from the device such as the Keyboard
Standard output	stdout	used to send output to a device such as a monitor or screen
Standard error	stderr	used to route the error message to the screen

There are two types of I/O functions in C programming, formatted and unformatted functions as shown in Figure 3.6.

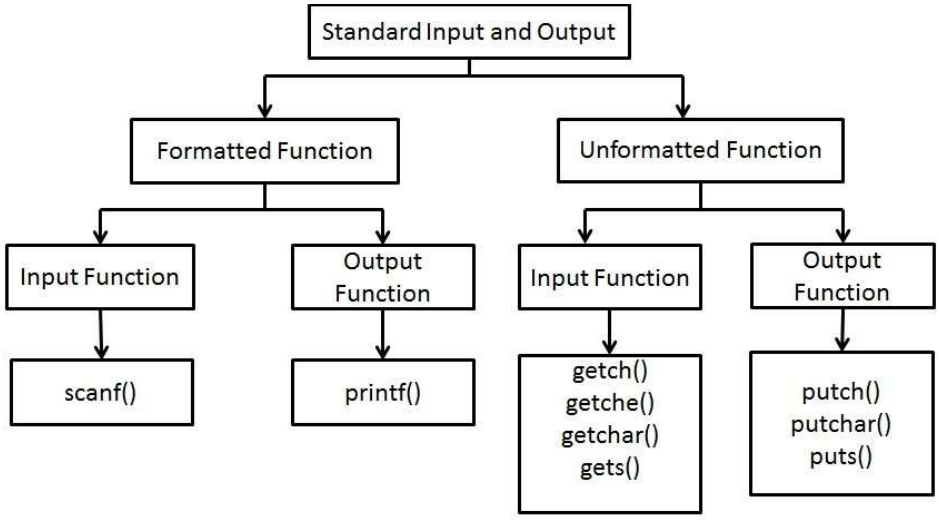


Figure 3.6 Formatted and Unformatted I/O Functions

Formatted I/O Functions: The two formatted functions in C are printf() and scanf();

`printf()`: It is a standard output function used for printing data on the standard output e.g.,

display screen. `printf()` can print the message, variable name, and a control string indicating the data type. The general syntax is given below.

```
int printf( const char *format, arg1, agr2....);
```

The first argument is the format string which is composed of ordinary characters and conversion specifications. This is followed by zero or more optional arguments.

```
Example: printf("welcome to c programming");
                int age = 20;
                printf(" student's age is %d", age);
```

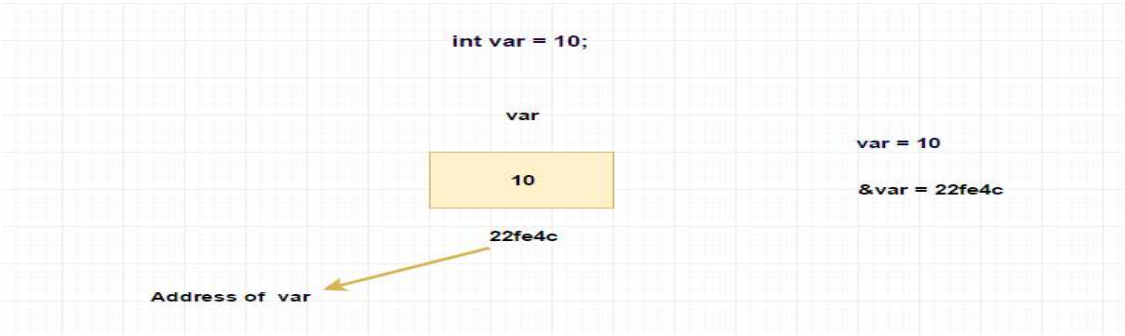
The scanf function is used for getting input. It contains the variable names, the address to store the value, and a format string. The general syntax is as follows.

```
scanf("<format specifier>" , <address of variable>);

Example: printf("enter student roll number);
         int rollno;
         scanf("%d", &rollno);
```

The address of a variable is given using & (Address Operator)

The address of a variable is obtained as shown below.



Unformatted Functions: The unformatted functions are `getchar(), putchar(), gets(), and puts()` as given below.

`getchar()` : This function is used to read character-type data from the input file stdin. It reads one character at a time till the user applies the enter key. The getchar() returns a character type. It reads the next character in the input file. The general syntax and examples are given below.

variable-name = `getchar();`

```
char ch;
ch = getchar();
```

`putchar()` : The `putchar()` function is used to display the character on the screen. The general syntax and an example are as follows.

```
putchar(variable-name);
char ch = 'C';
putchar(ch);
```

`gets()` : A set of characters are read using the gets() function. The gets() stops reading till the enter key is pressed or the end of the line is reached. The general syntax and the example are given below.

```
char *gets(char *str)
char ch[20];
gets(ch);
```

`puts()`: This function is used to print or display the data on the screen. The `puts()` stops when the end of the line of characters is reached. The general syntax and examples are shown below.

```
int puts(const char *str)
char name[20];
puts(name);
```

`gets()` and `puts()` work with a set of characters shown using array data type.
`getchar()` and `putchar()` are used for reading and printing a single character.

Every I/O function uses a format specifier or control string. The list of format specifiers used in C programming is as given in Table 3.9.

Table 3.9 Format Specifiers in C

Data type		**Format specifier**
Integer	short signed	%d or %I
	short unsigned	%u
	long singed	%ld
	long unsigned	%lu
	unsigned hexadecimal	%x
	unsigned octal	%o
Real	float	%f
	double	%lf
Character	signed character	%c
	unsigned character	%c
String		%s

4.3.1 Sample Programs Using I/O Functions

```
#include<stdio.h>
int main()
{
   int   rollno;
   float fees;
//Read input from the user and store it in memory
   scanf("%d%f",&rollno,&fees);
   printf("Roll Number = %d  Fees Paid = %f ",rollno,
     fees);
   return 0;
}
```

Program 3.3 C Program using I/O Functions

```
#include<stdio.h>
int main()
{
   long int phone_no;
   double cost;
//Read input from the user and store it in memory
   scanf("%ld%lf",&phone_no,&cost);
   printf("Phone Number = %ld  Amount Paid = %lf
      ",Phone_no, cost);
   return 0;
 }
```

Program 3.4 C Program using I/O Functions

```
#include <stdio.h>
int main()
{
   char c;
   printf("Enter some character...\n");
   c = getchar();
   printf("\n Entered character is: ");
   putchar(c);
   printf("\n")
   return 0;
}
```

Program 3.5 C Program using I/O Functions

In the above programs, \n is known as a newline character, and it is used to insert a blank or new line. These characters are commonly known as escape characters. The different escape characters are given in Table 3.11. The different escape characters will be discussed in the coming units.

Table 3.11 A Set of Escape Characters

Escape Sequence	Name	Meaning
\n	New Line	It moves the cursor to the start of the next line.
\r	Carriage Return	It moves the cursor to the start of the current line.
\t	Horizontal Tab	It inserts some whitespace to the left of the cursor and moves the cursor accordingly.
\f	Form Feed	It is used to move the cursor to the start of the next logical page.
\a	Alarm or Beep	It is used to generate a bell sound in the C program.
\b	Backspace	It is used to move the cursor backward.
\v	Vertical Tab	It is used to insert vertical space.
\\	Backlash	Use to insert backslash character.
\’	Single Quote	It is used to display a single quotation mark.

4.4. Self-Assessment Questions

Q1. Classify the operators in C programming. [8 marks, L1]
Q2. What are the logical operators? Mention the use with an application. [10 Marks, L2]
Q3. What are the bit-wise operators, explain using suitable I/O operations. [5 marks, L2]
Q4. Define the general syntax for printf() and scanf() functions [5 marks, L1]
Q5. What is a ternary operator? Explain how it can be used for checking conditions. [5 marks, L2]
Q6. Illustrate the purpose of cast, sizeof, and assignment operators using a suitable C Program. [10 marks, L3]
Q7. State the purpose of the format specifier in input/output functions. [5 marks, L1]
Q8. List the escape characters used for providing spaces using the C program. [5 marks, L2]
Q9. Illustrate the application of logical AND/OR operators using a suitable C Program. [10 marks, L3]
Q10. Classify the input/output functions in the C programming language. [5 marks, L1]

4.5 Self-Assessment Activities

A1. Develop a C program to convert temperature in Celsius to Fahrenheit. [5 marks, L4]
A2. Determine the output of the following C Program. [5 Marks, L5]

```
#include <stdio.h>
int main()
{
    int x = 1, 2, 3;
    printf("%d", x);
    return 0;
}
```

A3. Determine the output of the following C Program. [5 marks, L5]

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

```
int main()
{
    int x = (1, 2, 3);
    printf("%d", i);
    return 0;
}
```

A4. Write a C program to implement a simple calculator. [5 Marks, L2]

A5. Develop a C program to calculate the simple interest using the formatted I/O function. [5 Marks, L2]

4.6. Multiple-Choice Questions

Q1. The operator using only integers is ———. [1 mark, L1]

A. +
B. -
C. %
D. *

Q2. The output of the following program is ———. [1 mark, L1]

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

A. 1 and 1
B. 1 and 3
C. 2 and 3
D. 2 and 2

Q3. The highest precedence in C program is for _____ operators. [1 mark, L1]

A. Relational
B. Arithmetic
C. Logical
D. Ternary

Q4. The ——— operator of the following is the lowest precedence. [1 mark, L1]

A. %
B. /
C. ++
D. &&

Q5. The results in the following C Program on a 32-bit compiler is ———. [1 mark, L2]

```
#include <stdio.h>
int main()
{
    int k = 10;
```

```
        printf("%d and %d", sizeof(k),k);
        return 0;
    }
```

A. 2 and 10
B. 2 and 2
C. 4 and 10
D. Compile error

Q6. Output of the following C program is_____ [1 mark, L2]

```
#include <stdio.h>
int main()
{
    int x = 5, y = 10;
    z1 = y/x;
    z2 = y%x;
    printf("%d and %d", z1, z2);
    return 0;
}
```

A. 2 and 0
B. 0 and 2
C. Compiler error
D. 0 and 0

Q7. The results of logical expression in C Program are____ [1 mark, L1]

A. T or F
B. Yes or No
C. 1 or 0
D. None of the above

Q8. The expression `4+6/3*2-2+7%3` evaluates to_____ [L1 mark, L3]

A. 6
B. 7
C. 4
D. 1

Q9. The statement `printf("%d", ++6);` will print——— [1 mark, L3]

A. 7
B. 6
C. Garbage Value
D. Compilation error

Q10. Mention the output of the following C program ———. [1 mark, L2]

```
int main(void)
{
   int x = 10.3 % 2;
   printf("Value of x is %d", x);
   return 0;
 }
```

A. Compilation error
B. Value of x is 10
C. Value of x is 3
D. Value of x is 0.4

4.7. Keys to Multiple-Choice Questions

Q1. % (C).
Q2. 2 and 3 (C).
Q3. Arithmetic (B).
Q4. && (D).
Q5. 2 and 10 (A).
Q6. 2 and 0 (A).
Q7. 1 or 0 (C)
Q8. 7 (B)
Q9. Compilation error (D)
Q10. Compilation error (A)

4.8. Summary of the Unit

In every C program, the data input and output are handled using formatted and unformatted I/O functions. The printf() and scanf() are formatted I/O functions. The getchar(), putchar(), gets() and puts() functions are used for handling single-character data. The operations on the operands are performed using operators. In C programming, operators are classified based on their functionality. Every expression of operators and operands is evaluated using precedence and associativity rules.

4.9. Keywords:

- Operators
- Escape Characters
- Ternary Operator
- Bitwise Operator
- Sizeof operator
- Arithmetic, relational, and Logical Operators
- Precedence and Associativity
- Formatted and unformatted Input Output Functions
- printf()
- scanf()

4.10. Recommended Learning Resources

[1] Herbert Schildt. (2017). *C Programming: The Complete Reference*. 4th ed. USA: McGraw-Hill.