

# **Programming Using C**

## **Week 02**

---

### **Operators and Expressions, Managing IO Operations**

**Prepared By: REC Faculty 4.0 Team**

# TOPICS

- 
- ❑ INTRODUCTION- OPERATORS AND EXPRESSIONS
  - ❑ PRECEDENCE
  - ❑ ASSOCIATIVITY
  - ❑ TYPES OF OPERATORS
  - ❑ TYPES OF EXPRESSIONS
  - ❑ ARITHMETIC OPERATORS
  - ❑ RELATIONAL OPERATORS
  - ❑ LOGICAL OPERATORS
  - ❑ ASSIGNMENT OPERATORS
  - ❑ AND USAGE OF I/O FUNCTIONS

# INTRODUCTION- OPERATORS

---

- **An operator is a symbol that operates on a value or a variable. For example: + is an operator to perform addition.**
- Operators are used in programs to manipulate data and variables.
- They usually form a part of the mathematical or logical expressions. C has a wide range of operators to perform various operations.

# INTRODUCTION- EXPRESSIONS

---

- An expression is a formula in which operands are linked to each other by the use of operators to compute a value.
- An operand can be a function reference, a variable, an array element or a constant.
- Example:
  - $a+b/c$

## **TYPES OF OPERATORS**

**Arithmetic operators**  
**Relational operators**  
**Logical operators**  
**Assignment operator**  
**Increment and decrement operators**  
**Address of operator**  
**Conditional (or) ternary operator**  
**Bitwise operators**  
**Comma operator**

## **TYPES OF EXPRESSIONS**

Arithmetic expressions  
Relational expressions  
Logical expressions  
Conditional expressions

# OPERATOR PRECEDENCE

---

- **Operator precedence:** It decides the order of evaluation of operators in an expression. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator, that is if you have an expression like this,
- Example:
- $a = 5 + 8 * 2$
- •  $8 * 2$  gets evaluated first, then the result (16) will be added to 5. Final result will be 21.

# OPERATOR ASSOCIATIVITY

---

- **Associativity:** It defines the order in which operators of the same precedence are evaluated in an expression. Associativity can be either from left to right or right to left. In C, each operator has a fixed priority or precedence in relation to other operators.
- Example:
- $C=1+2+3$  for  $+$  op associativity is left to right, hence  $1+2$  will be evaluated first, then  $+3$  with previously computed value.

Operator	Description	Associativity
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to or not equal to	left to right
&&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^=  = <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	comma operator	left to right

# ARITHMETIC OPERATORS AND EXPRESSIONS

OPERATOR	DESCRIPTION
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo Division(Reminder after Integer Division)

Arithmetic Expression is,

$$c=a+b*d$$

which involves arithmetic operators in it.

# Important Points:

- For '/' Operator if both the operands are integers then fractional part is truncated. If either one or both the operands are float then fractional part is not truncated.
- 
- Eg:  $a=5.0$   $b=5$   $c=2$ , then  $a/c=2.500000$  and  $b/c=2$
  - '%' cannot be applied on float and double variable.
  - + and - have equal precedence.
  - /, \* and % have equal precedence but higher precedence than + and -.
  - Unary + and - have higher precedence than / and \*.
  - Arithmetic operators associate left to right.

## Example

```
int main()
{
float a=7/2;
printf("%f",a);
return 0;
}
```

---

□ OUTPUT:3.000000

□ What will be the output if u change the constant 7 to 7.0 or 2 to 2.0?

**3.500000**

□ What will be the output if u change datatype of 'a' from float to int?

**3**

# Example

Tell the output

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

```
int main()
```

```
{
```

```
    float a=5%2;
```

```
    printf("%f",a);
```

```
    return 0;
```

```
}
```

**OUTPUT:1.000000**

What will be the output if u change the constant 5 to 5.0 or 2 to 2.0?

**Error: invalid operands to binary %**

# RELATIONAL OPERATORS AND EXPRESSIONS

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

OPERATOR	DESCRIPTION
>	Greater than
>=	Greater than and equal to
<	Lesser than
<=	Lesser than and equal to
==	Equal to
!=	Not equal to

Relational Expression is,  
 $a >= b$ , which involves relational operators

# Important points:

- $>$  ,  $>=$  ,  $<$  ,  $<=$  all have the same precedence. Just below them in precedence are the equality operators:  $==$   $!=$ .
- Relational operators have lower precedence than arithmetic operators.
- Associativity: left to right
- **Example:**
  - If  $a=10, b=10$  then
    - $a > b$  will be false(0)
    - $a >= b$  will be true(1)
    - $a < b$  will be false(0)
    - $a <= b$  will be true(1)
    - $a == b$  will be true(1)
    - $a != b$  will be false(0)

## Example:

2. What will be the value of i?

```
int i=16;
```

---

```
i=i>15+7;
```

OUTPUT: 0

3. What will be the value of i?

```
int i=16;
```

```
i=-i>15;
```

OUTPUT:0

## Example:

---

□ `int t= 10>12>13>=14`

evaluates to **0**

□ `int x = 41 >12 >=1>=1`

evaluates to **1**

# LOGICAL OPERATORS AND EXPRESSIONS

---

- The logical operators are used when we want to test more than one condition and make decisions. The logical operators are

OPERATOR	DESCRIPTION	No of operands
&&	Logical AND	2 operands( a && b)
	Logical OR	2 operands(a   b)
!	Logical NOT	1 operand (!a)

# LOGICAL OPERATORS AND EXPRESSIONS

Operand 1	Operand 2	Operand 1 && Operand 2	Operand 1    Operand 2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Operand	!Operand
0	!0=1
Non-zero (for example)	!5=0

# Important Points

- Logical Expression is,
- $(a-b) \&\& b \parallel c$ , which involves logical operators.

- **Important Points:**

---

- The precedence of  $\&\&$  is higher than that of  $\parallel$ , and both are lower than relational and equality operators.
- Logical not has higher precedence than  $\&\&$ ,  $\parallel$ , arithmetic and relational operators.
- Associativity: Left to Right.
- Logical AND ( $\&\&$ ) operator evaluates the second expression only when the first expression is true. It returns a value true only if all the expressions are true.
- Eg:  $12 > 11 \&\& 12 > 10$  returns true.

# Important Points

- Eg:  $10 > 12 \ \&\& \ 12 > 11$  (First expression is false hence it will not evaluate the second expression, it will return false).

---

- Logical OR ( $\parallel$ ) evaluates the second expression only when the first expression is false. It returns true if any one of the expression is true.
- Eg:  $12 > 11 \parallel 12 > 10$  (First expression is true hence it will not evaluate the second expression, it will return true).
- Eg:  $10 > 12 \parallel 12 > 11$  (returns true after evaluating both the expressions)

# Important Points

Example:

1. What will be the value of i?
- 

```
int i=16;
```

```
i=i>15 && 7;
```

OUTPUT: 1

2. What will be the value of i?

```
int i=16;
```

```
i=!i>15;
```

OUTPUT:0

# ASSIGNMENT OPERATOR

- Assignment operator is used to assign the result of an expression to a variable. The most commonly used assignment operator is '='.Examples are,
  - num = 25;
  - age = 18;
  - pi = 3.14;
  - area = 3.14 \* r \* r;
- **Important points**
  - 1. It has a precedence lower than most of all the operators except comma.
  - 2. Associativity: Right to Left

# ASSIGNMENT OPERATOR

---

## Example:

- ❑ Point out the errors, if any, in the following C statement,
- ❑ `a=b=3=3;`    **constant cannot be assigned to another constant**
- ❑ `a=b=c=3;`**no error**
- ❑ What is the value of a in `a=b=c=9;`    **VALUE=9**

# SHORTHAND ASSIGNMENT OPERATOR

---

C has a set of 'shorthand' assignment operators.

## Syntax

variable operator= expression;

is equivalent to

variable = variable operator (expression);

**Example** Simple Assignment Operator Shorthand Operator

n = n + 1      n += 1

n = n - 1      n -= 1

n = n \* 1      n \*= 1

# SHORTHAND ASSIGNMENT OPERATOR

---

C has a set of 'shorthand' assignment operators.

## Syntax

variable operator= expression;

is equivalent to

variable = variable operator (expression);

**Example** Simple Assignment Operator Shorthand Operator

n = n + 1      n += 1

n = n - 1      n -= 1

n = n \* 1      n \*= 1

# TOPICS

---

- BITWISE OPERATORS
- COMMA OPERATOR
- INCREMENT AND DECREMENT OPERATORS

# BITWISE OPERATORS

- One of C's powerful features is a set of bit manipulation operators.
- This permits the programmer to access and manipulate individual bits within a piece of data.
- The bitwise operator can **operate upon int and char data types** but not on float and double data types.
- All operators except ~ operator are binary operators which requires two operands.
- While using the bit operators, **each operand is treated as a binary number** consisting of a series of individual 1s and 0s.
- The respective bits in each operand are then compared on a bit by bit basis and result is determined based on the selected operation.

# BITWISE OPERATORS

OPERATOR	MEANING
&	BITWISE AND
	BITWISE OR
^	BITWISE EXCLUSIVE OR
<<	SHIFT LEFT
>>	SHIFT RIGHT
~	ONE'S COMPLEMENTT

# BITWISE OPERATORS

A	B	A&B	A B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Example

A	B	A&B	A B	A^B
11	01	01	11	10

# SHIFT OPERATORS:LEFT SHIFT

## BITWISE OPERATORS

### << Shift Left

<u>SYNTAX</u>	<u>BINARY FORM</u>	<u>VALUE</u>
x = 7;	00000111	7
x=x<<1;	00001110	14
x=x<<3;	01110000	112
x=x<<2;	11000000	192

We can also use a formula to find the answer,

$$x \ll y = x * 2^y$$

# SHIFT OPERATORS:RIGHT SHIFT

## BITWISE OPERATORS

### >> Shift Right

<u>SYNTAX</u>	<u>BINARY FORM</u>	<u>VALUE</u>
x = 192;	11000000	192
x=x>>1;	01100000	96
x=x>>2;	00011000	24
x=x>>3;	00000011	3

We can also use a formula to find the answer,

$$x \gg y = x / 2^y$$

# SHIFT OPERATORS:RIGHT SHIFT

What will be the output of the C program?

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

---

```
int main()
```

```
{
```

```
    int a = 2,b = 5;
```

```
    a = a^b;
```

```
    b = b^a;
```

```
    printf("%d %d",a,b);
```

```
    return 0;
```

```
}
```

• **OUTPUT 7 2**

• ANSWER : D

# SHIFT OPERATORS

What will be the output of the C program?

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

---

```
int main()
```

```
{
```

```
    int num = 8;
```

```
    printf ("%d %d", num << 1, num >> 1);
```

```
    return 0;
```

```
}
```

• **OUTPUT 16 4**

# COMMA OPERATOR:

---

- The comma operator can be used to link the related expressions together. A comma-linked **list of expressions are evaluated left to right** and the **value of right most expression is the value of the combined expression**.
- **Example:**
- **`sum = (a = 10, b = 20, a + b);`**
- The statement first assigns the value 10 to a, then assigns 20 to b, and finally assigns 30 (i.e.  $10 + 20$ ) to sum. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

# COMMA OPERATOR:

What will be the output?

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

---

```
int main()
{
    int a = 5;
    a = (1, 2, 3);
    printf("%d", a);
    return 0;
}
```

**OUTPUT:3**

# COMMA OPERATOR:

---

**The following expression `a=(a=9,b=3,c=a+b)` evaluates to .....**

**12**

# COMMA OPERATOR:

**Point out the error if present , otherwise the output**

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

---

```
int main()
```

```
{
```

```
    int a = 5,b,c;
```

```
    a = a=9, b=3, c=a+b;
```

```
    printf("%d", a);
```

```
    return 0;
```

```
}
```

**NO ERROR OUTPUT=9**

# INCREMENT AND DECREMENT OPERATORS

---

- Increment Operators are used to increase the value of the variable by one.
- Decrement Operators are used to decrease the value of the variable by one.
- Both increment and decrement operator are used on a single operand and variable, so it is called as a unary operator.
- **This operators cannot be applied on constants.**

# INCREMENT AND DECREMENT OPERATORS

---

- Two kinds of Increment /Decrement Operators
- Prefix/Postfix
- Prefix operator first increments / decrements and then makes the assignment.
- Postfix operator makes the assignment and then increments/decrements the value

# INCREMENT AND DECREMENT OPERATORS

Operator	Expression	Description	Example
++	++x	Pre-Increment	If x=5, then y=++x makes x=6 and y=6
	x++	Post-Increment	If x=5, then y=x++ makes y=5 and x=6
--	--x	Pre-Decrement	If x=5, then y=--x makes y=4 and x=4
	x--	Post-Decrement	If x=5, then y=x-- makes y=5 and x=4

# INCREMENT AND DECREMENT OPERATORS

---

- Increment operator has higher precedence than decrement operator.
- Precedence of postfix ++ is higher than prefix ++ and their associativity is also different.
- Associativity of prefix ++ is right to left and postfix ++ is left to right.

# INCREMENT AND DECREMENT OPERATORS

## Example:

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

```
int main()
```

```
{
```

```
int x,i;
```

```
i=10;
```

```
x=++i;
```

```
printf("x: %d %d",x,i);    //prints x=11  i=11
```

```
x=i++;
```

```
printf("x: %d %d",x,i);    //prints x=11  i=12
```

```
x=--i;
```

```
printf("x: %d %d",x,i);    //prints
```

```
x=11  i=11
```

```
x=i--;
```

```
printf("x: %d %d",x,i);    //prints
```

```
x=11  i=10
```

```
}
```

# TOPICS

---

□ **CONDITIONAL OPERATOR**

□ **ADDRESS OF OPERATOR**

# CONDITIONAL OPERATOR (TERNARY OPERATOR)

---

- The conditional operator (?) and (:) are sometimes called ternary operators because they take three arguments.
- **Syntax:**
  - **Expression 1? (Expression 2): (Expression 3)**
- Expression-1 is evaluated first.
- If it is true, then the expression-2 is evaluated and becomes the result value.
- If expression-1 is false (zero), expression-3 is evaluated and its value becomes result value
- Note that only one of the expressions (either expression-2 or expression-3) is evaluated..

# CONDITIONAL OPERATOR (TERNARY OPERATOR)

---

- Example:

a=10, b=11

d=(a>b)? a:b

- **VALUE OF d=11**

# ADDRESS OF OPERATOR:

---

- Ampersand (&) is the “address of” operator. It is used to fetch the memory address of a variable.
- Generally used in scanf() statement and in pointer concept.
- **Example:**
- **scanf(“%d %d”,&a,&b);**

# MCQS

---

**What will be the output?**

```
#include <stdio.h>
int main()
{
    int A = 10, j, nu=9;
    j = nu < 0 ? 0 : nu - A;
    printf("%d", j);
    return 0;
}
```

**OUTPUT: -1**

# MCQS

---

**What will be the output of the following programs?**

```
# include <stdio.h>
int main()
{
int k, num = 30;
k = (num > 5 ? (num <= 10 ? 100 : 200) : 500);
printf("%d", k);
return 0;
}
```

**OUTPUT: 200**

# MCQS

---

What will be the output? [N.B:- .2f is used to print upto 2 decimal places of a floating point number]

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

```
int main()
```

```
{
```

```
    float a = 9.0;
```

```
    printf("Ans = %.2f", (9 / 5) * a + 10);
```

```
    return 0;
```

```
}
```

**OUTPUT: Ans=19.0**

# MCQS

---

**What is the output of the following C code?**

```
#include <stdio.h>
int main()
{
    int var = 0101;
    var = var + 5;
    printf("%d", var);
    return 0;
}
```

**OUTPUT: 70**

# MCQS

---

What will be the output of the following C code?

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

```
int main()
```

```
{
```

```
    int x = 2, y = 1, z = 9;
```

```
    x = x && y || z;
```

```
    printf("%d", x || !y && z);
```

```
    return 0;
```

```
}
```

**OUTPUT: 1**

# MCQS

---

What will be the output of the following C code?

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

```
int main()
```

```
{
```

```
    int i = 0, j = 1, k = 2, m;
```

```
    m = i++ || j++ || k++;
```

```
    printf("%d %d %d %d", m, i, j, k);
```

```
    return 0;
```

```
}
```

**OUTPUT: 1 1 2 2**

# MCQS

---

What will be the output of this program on an implementation where int occupies 2 bytes?

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

```
int main()
```

```
{
```

```
    int i = 3;
```

```
    int j;
```

```
    j = sizeof(++i + ++i);
```

```
    printf("i = %d j = %d", i, j);
```

```
    return 0;
```

```
} OUTPUT: i=3 j=4
```

# PROGRAM STATEMENT 1

---

- Suzie came over to Monk's place today with a box of Monk's favorite cookies! But Monk has to play a game with her in order to win the cookies. Suzie takes out 10 balls from her bag, each ball having an integer on it between 0 to 9. She places all these balls in a box and closes it. The game starts by Monk choosing an integer from 0 to 9. It is called as Monk's chosen integer. Post this, Suzie draws 1 ball from the box **randomly** and notes the integer on it. The rule is simple, Monk wins if Monk's chosen is same as what suzie draws. Print 1 if monk wins otherwise print 0

# PROGRAM STATEMENT 1

---

```
#include<stdio.h>
int main()
{
    int m_no;
    int s_no;
    scanf("%d %d",&m_no,&s_no);
    int j=(m_no==s_no)?printf("Monk Win"):printf("suzie Win");
    return 0;
}
```

```
C:\TDM-GCC-64\bin>gcc monk.c
```

```
C:\TDM-GCC-64\bin>a
```

```
9
```

```
0
```

```
suzie Win
```

```
C:\TDM-GCC-64\bin>a
```

```
9
```

```
9
```

```
Monk Win
```

```
C:\TDM-GCC-64\bin>
```

# TOPICS

---

- **INTRODUCTION**
- **CONSOLE I/O FUNCTIONS**
- **FORMATTED I/O FUNCTIONS**
- **UNFORMATTED I/O FUNCTIONS**
- **MCQS**
- **PROBLEMS**

# INTRODUCTION

---

- C Language has a set of library functions to perform I/O operations.
- The I/O library functions are listed the “header” file <stdio.h>.
- There are numerous library functions available for I/O. These can be classified into two broad categories:
- (a) **Console I/O functions** - Functions to receive input from keyboard and write output to VDU.
- (b) **File I/O functions** - Functions to perform I/O operations on floppy disk or hard disk.

# CONSOLE-I/O FUNCTIONS

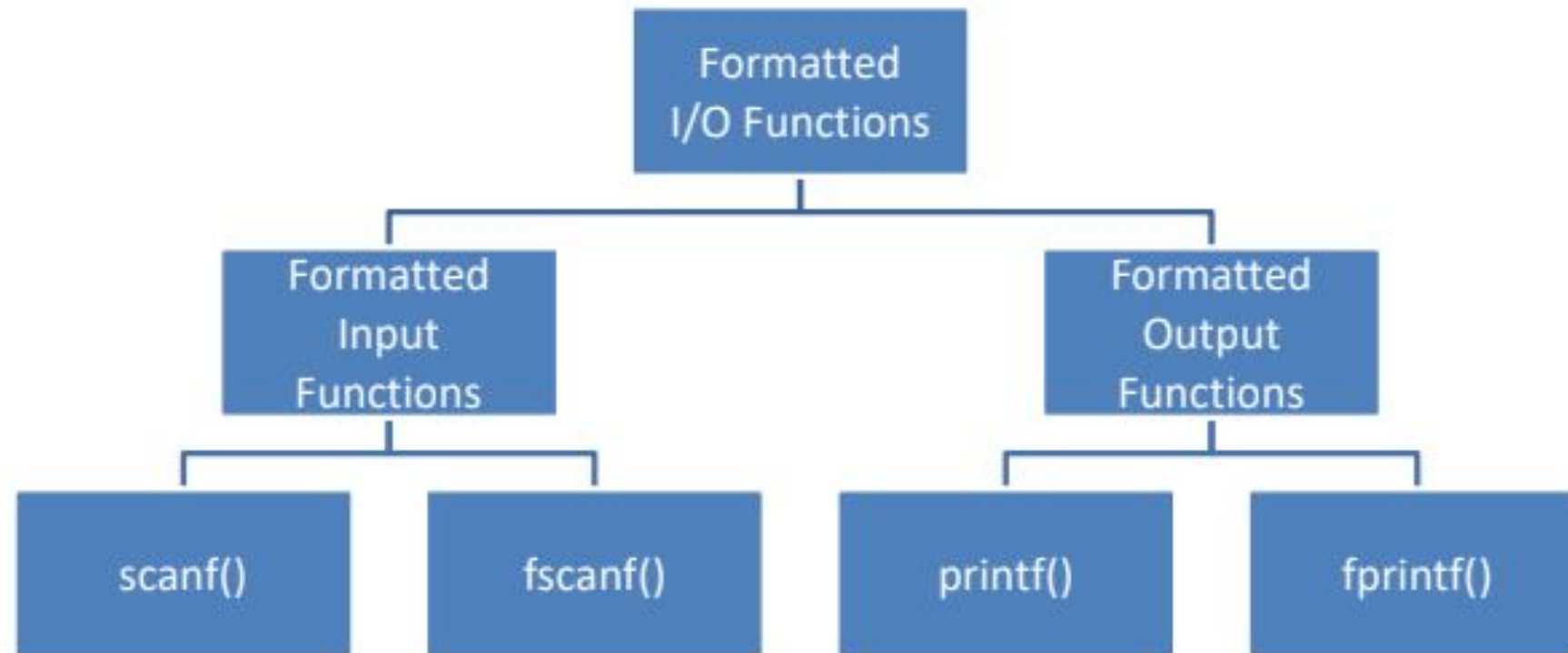
---

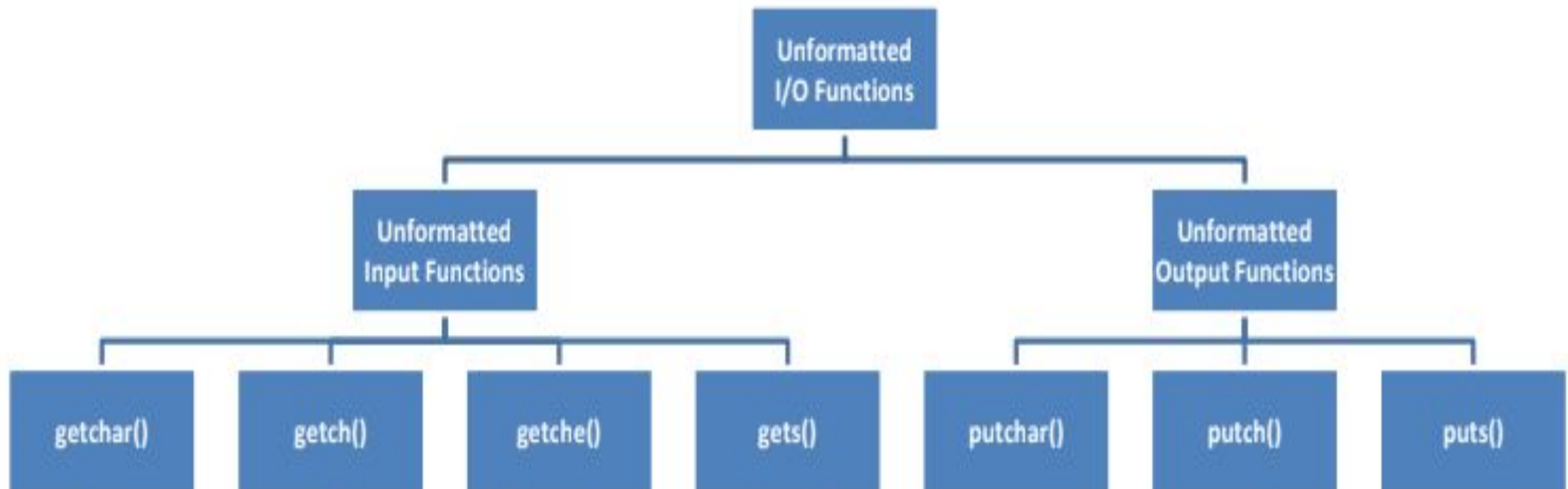
- The screen and keyboard together are called a console.
- Console I/O functions is further classified into two categories,
  - **formatted console I/O functions.**
  - **unformatted console I/O functions.**

# CONSOLE-I/O FUNCTIONS

---

- **Formatted functions** allow to use the input got from keyboard and output displayed on VDU as per our requirements.
- For example, if values of average marks and percentage marks are to be displayed on the screen, then the details like where this output would appear on the screen, how many spaces would be present between the two values, the number of places after the decimal points, etc. can be controlled using formatted functions.
- **In Unformatted functions** ,no such things are taken into account.





# FORMATTED CONSOLE-I/O FUNCTIONS-printf() function

---

- This function provides formatted output to the screen.
- Syntax:
- **int printf (“format”, var1, var2, ...var n);**
- The “format” includes a listing of the data types(format specifier) of the variables to be output and, optionally, some text and escape sequences.
- The return type of printf() is “int” and it returns *total number of Characters Printed, Or negative value if an output error or an encoding error*
- Example:
- `int b=10;`
- `printf (“The value of b is: %d \n”,b);// %d –format specifier of integer`

# FORMATTED CONSOLE-I/O FUNCTIONS-printf() function-format specifiers

**Objective-C Data Types Table**

Type	Example	Specifier
char	'a', 'O', '\n'	%c
int	14, -14, 780, 0xEEC0, 098	%i, %d
unsigned int	10u, 121U, 0xEFu	%u, %x, %o
long int	18, -2100, 0xfeefL	%ld
unsigned long int	13UL, 101ul, 0xfefeUL	%lu, %lx, %lo
long long int	0xe5e5e5LL, 501ll	%lld
unsigned long long int	10ull, 0xffeeULL	%llu, %llx, %llo
float	12.30f, 3.2e-5f, 0x2.2p09	%f, %e, %g, %a
double	3.1415	%f, %e, %g, %a
long double	3.5e-5l	%Lf, %Le, %Lg, %La
id	Nil	%@

# FORMATTED CONSOLE-I/O FUNCTIONS-printf() function-optional specifiers

---

Specifier	Description
dd	Digits specifying field width
.	Decimal point separating field width from precision (precision stands for the number of places after the decimal point)
dd	Digits specifying precision
-	Minus sign for left justifying the output in the specified field width

# printf() function-optional specifiers

- The field-width specifier is used to decide how many columns should be used on the screen to print a value.
- Eg: %10d, prints the variable as a decimal integer in a field of 10 columns.
- If the value of variable is not up to fill the entire field then the value is right justified and padded with blanks on left.

- Eg: int a=115,

- `printf("%10d" a)`.

							1	1	5
--	--	--	--	--	--	--	---	---	---

- If the field specifier includes a minus then value is left justified and padded with blanks on right. If the field width is less than what is required then it is completely ignored.

# printf() function-Example

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

```
int main()
```

```
{
```

```
int weight = 63 ;
```

```
printf ( "\nweight is%2d", weight ) ;
```

```
printf ( "\nweight is%3d", weight ) ;
```

```
printf ( "\nweight is%4d", weight ) ;
```

```
printf ( "\nweight is%5d", weight ) ;
```

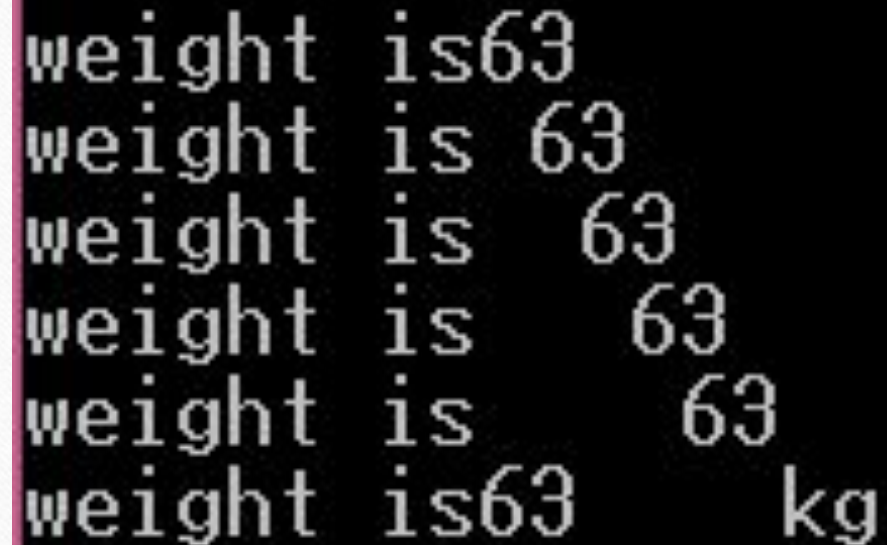
```
printf ( "\nweight is%6d", weight ) ;
```

```
printf ( "\nweight is%-6dkg", weight ) ;
```

```
return 0;
```

```
}
```

**OUTPUT:**



```
weight is63
weight is 63
weight is  63
weight is   63
weight is    63
weight is     63 kg
```

# printf() function-Example

```
#include <stdio.h>
int main()
{
    printf ( "\n%10.2f%10.2f", 5.0, 13.5) ;
    printf ( "\n%10.2f%10.2f ", 305.0, 1200.9);
    return 0;
}
```

## OUTPUT:



```
      5.00      13.50
    305.00    1200.90
C:\TDM-GCC-64\PL-Phase 1>
```

# printf() function-Example

```
#include <stdio.h>

int main()
{
    char ch = 'z' ;
    int i = 125 ;
    float a = 12.55 ;
    printf ( "\n%c %d %f", ch, ch, ch ) ;
    printf ( "\n%c %d %f",i ,i, i ) ;
    printf ( "\n%f %d\n", a, a ) ;
}
```

## OUTPUT:

```
z 122 garbage value
} 125 garbage value
12.550000 garbage value
```

# INPUT/OUTPUT FUNCTIONS

---

## FORMATTED INPUT FUNCTION: scanf ()

This function provides for formatted input from the keyboard.

### Syntax:

**int scanf ( “format” , &var1, &var2, ...) ;**

- The “format” is a listing of the data types(format specifier) of the variables to be input and the & in front of each variable name tells the system WHERE(address) to store the value that is input. It provides the address for the variable.
- **It returns total number of Inputs Scanned successfully, or EOF if input failure occurs before the first receiving argument was assigned.**

### Example:

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

# USAGE OF FORMAT SPECIFIER IN scanf() function

- **Example:**

**// Extracting single digit using C Code.**

---

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

```
int main()
```

```
{ //1234
```

```
int a,b,c,d;
```

```
scanf("%1d%1d%1d%1d", &a, &b, &c,  
&d);
```

```
printf("%d", a+b+c+d);
```

```
return 0;
```

```
}
```

**OUTPUT:**

**1234**

**10**

# USAGE OF FORMAT SPECIFIER IN `scanf()` function

- **Example:**

// getting digit input in expression  
(a+b) format using C Code.

---

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

```
int main()
```

```
{
```

```
int a;
```

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

```
printf("%d %d", a, b);
```

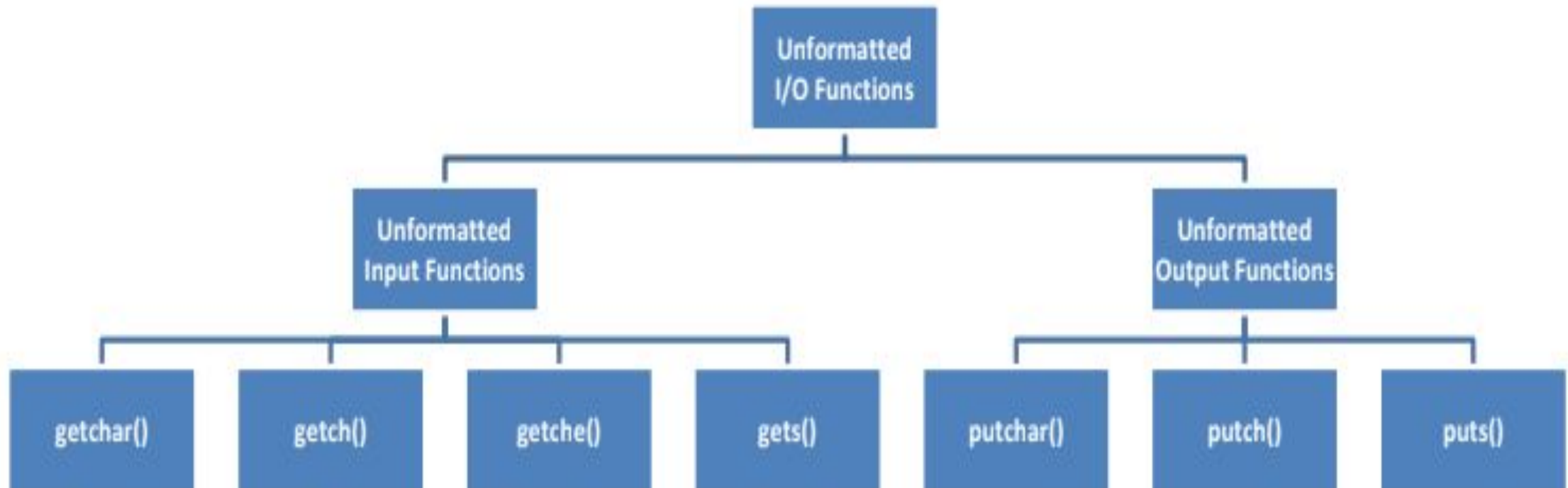
```
return 0;
```

```
}
```

**OUTPUT:**

10+11

10 11



# UNFORMATTED FUNCTIONS

## getchar(), getch() and getche()

- All of these functions read a character from input and return an integer value.
- **getchar()** It reads a single character from the standard input and returns the corresponding integer value (typically ASCII value of read character) on success. It returns EOF on failure.

**Syntax:**

**int getchar();**

```
#include <stdio.h>
int main()
{
    int c=getchar();
    printf("%c",c);
}
```

# UNFORMATTED FUNCTIONS

**getch():** getch() is a nonstandard function and is present in conio.h header file which is mostly used by MS-DOS compilers like Turbo C. It is not part of the C standard library or ISO C.

- Like above functions, it reads also a single character from keyboard. But it does not use any buffer, so the entered character is immediately returned without waiting for the enter key.

Syntax:

```
int getch();
```

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

```
int main()
```

```
{
```

```
    printf("%c", getch());
```

```
    return 0;
```

```
}
```

**getche()** is similar to getch();

# UNFORMATTED FUNCTIONS

---

**gets():** Reads characters from the standard input (stdin) and stores them as a C string into str until a newline character or the end-of-file is reached.

## Syntax:

`char * gets ( char * str );`

str : Pointer to a block of memory (array of char)

where the string read is copied as a C string.

returns : the function returns str.

```
int main()
{
    char buf[50];
    printf("Enter a string: ");
    gets(buf);
    printf("string is: %s\n", buf);

    return 0;
}
```

# UNFORMATTED FUNCTIONS

**The putchar(int char)** :is used to write a character, of unsigned char type, to stdout. This character is passed as the parameter to this method.

## Syntax:

**int putchar(int ch);**

This function returns the character written on the stdout as an unsigned char. It also returns EOF when some error occurs.

```
#include <stdio.h>

int main()
{
    // Get the character to be written
    char ch = 'G';
    // Write the Character to stdout
    putchar(ch);
    return (0);
}
```

# UNFORMATTED FUNCTIONS

**puts() function** is used to write a line to the output screen. In a C program, we use puts function as below.

## Syntax:

```
int puts(const char *string)
```

string – data that should be displayed on the output screen.

If successful, non-negative value is returned. On error, the function returns EOF.

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

int main()
{
    char string[40];
    scanf("%s",string);
    puts(string);
    return 0;
}
```

# MCQS

**OUTPUT:**

**REC Hello REC 3**

What will be the output?

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

```
int main()
```

```
{
```

```
    printf("Hello ", printf("REC "));
```

```
    int i=printf("REC");
```

```
    printf(" %d",i);
```

```
    return 0;
```

```
}
```

```
.
```

# MCQS

What will be the output?

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

```
int main()
```

```
{
```

```
    int j=printf("Hello", printf("REC "));
```

```
    int i=printf("REC");
```

```
    printf(" %d %d",j,i);
```

```
    return 0;
```

```
}
```

**OUTPUT:**

**REC Hello REC 5 3**

# MCQS

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

```
int main()
```

```
{
```

```
int i = 1, j = 4;
```

```
printf("%d %d %d", i, j);
```

```
return 0;
```

```
}
```

a) Compile time error

b) 1 4

c) 1 4 some garbage value

d) Undefined behaviour

**OUTPUT:**

Answer: c

# MCQS

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

```
int main()
```

```
{
```

```
int i = 1, j = 4, k = 1;
```

```
printf("%d %d ", i, j, k);
```

```
return 0;
```

```
}
```

a) Compile time error

b) 1 4

c) 1 4 some garbage value

d) Undefined behaviour

**OUTPUT:**

Answer: b

# MCQS

What will be the output of the C program?

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

```
int main()
```

---

```
{
```

```
    printf("%d",printf("rec"));
```

```
    return 0;
```

```
}
```

A. compilation error

B. Runtime error

C. rec

D. rec3

**OUTPUT:**

Answer:D

# MCQS

```
int main()
{
    int a = 4;
    printf("hello"+3);
    return 0;
}
```

- A. compilation error
- B. hellohellohello
- C. hello
- D. lo

**OUTPUT:**

Answer:D