Using const Keyword

Syntax:

```
const contant_type constant_name = value;
```

Or

```
constant_type const const_name = value;
```

Example:

```
int const a = 1;
const int a =2;
```

You can use the const keyword before or after the type.

➢ **Example:**

```
#include<stdio.h>
int main()
{
const float PI=3.14;
int radius = 2;
float area = PI * radius * radius;
printf("The area  of circle is: %f",area);
return 0;
}
```

Output:

```
The area  of circle is: 12.560000
```

■ VOLATILE VARIABLE

A volatile variable is the one whose values may be changed at any time by some external sources.Volatile in is a keyword which is used with variables to inform the compiler not to apply any optimizations to code dealing with the variable.Syntax:

```
volatile <data_type><variable_name>;
```

// or

```
<data_type> volatile <variable_name>;
```

For example, the variable can be declared volatile as follow:

```
volatile int date;
```

This indicates that the value of variable date may be altered by some external factors. When we declare a variable as volatile the compiler will examine the value of the variable each time it is encountered to see if an external factor has changed the value. We can also change the value of the variable declared as volatile in the program. If we wish that the value of the variable must not be modified by the program while it may be altered by some other process then we may declare the variable as both constant and volatile. For example:

```
volatile const int location = 100;
```

2.8.5 Type Conversion

If any expression involves two different types of operands, then to evaluate expression properly both operands are converted into same type. Such conversion is called **Type Conversion.**

- **Implicit Type Conversion**

Whenever as expression involves two different types of operands, C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance. This automatic conversion is known as **Implicit Type Conversion**. If the operands are of different types, then the operand with a lower type is upgraded to the higher type and then the operation is performed.

➢ **Example:**

```
#include<stdio.h>
void main()
{
    int a=5,b;
    float c=5.5;
    b=a+c;
printf("b = %d",b);
}
```

Output:

```
b = 10
```

Here, a will be upgraded to float because another operand c is float. So a+c is evaluated to 10.5 and assigned to b, which is integer, so truncated value of 10.5 will be the value of b. So, 10 will be assigned to b.

- **Explicit Type Conversion (Type Casting)**

Type casting is explicitly specified by the programmer in the program.

Syntax:

```
(datatype) expression;
```

Where the datatype is to which the expression value is converted.

Example:

```
#include<stdio.h>
void main()
{
        int a;
        float b=5.5;
        a=int(b);
printf("a = %d",a);
}
```

Output:

```
a = 5
```

Here, the value of b (5.5) is converted to integer by truncation.

2.9 INTRODUCTION OF DIFFERENT TYPES OF OPERATORS

Operators are symbols that help in performing operations of mathematical and logical nature. The classification of C operators are as follows:

- Arithmetic
- Relational
- Logical
- Bitwise
- Assignment
- Conditional
- Special

2.9.1 Arithmetic Operators

These operators perform arithmetic or mathematical operations like addition (+), subtraction (-), multiplication (*), division (/), the remainder of the division (%), increment (++), decrement (–).

There are two types of arithmetic operators:

- **Unary Operators:** This type of operator works with a single value (operand) like ++ and –.
- **Binary Operators:** This type of operator works with two operands like +,-,*,/

Operator	Function
+	Adds two values
–	Subtract a second value from first
*	Multiply two values
/	Divide numerator by the denominator
%	Remainder of division
++	Increment operator – increases integer value by one.
—	Decrement operator – decreases integer value by one

➤ **Example :** C Program using arithmetic operators

```
#include <stdio.h>
int main()
{
int a = 12, b = 6, c;
c = a + b;
printf("a+b = %d \n", c);
c = a - b;
printf("a-b = %d \n", c);
c = a *b;
```

```
printf("a*b = %d \n", c);
c = a / b;
printf("a/b = %d \n", c);
c = a % b;
printf("Remainder when a divided by b = %d \n", c);
return 0;
}
```

Output :

```
a+b = 18
a-b = 6
a*b = 72
a/b = 2
```

Remainder when a divided by b=0

The Increment and Decrement Operators :

++ (Increment) operator adds one to operand. val++; is the same as val = val + 1;

— (Decrement) operator subtracts one from operand. val—; is the same as val = val - 1;

It is also possible to use ++i and —i instead of i++ and i—. However, these two forms have an important difference. Consider these examples:

Example1 (Postfix)	**Example2 (Prefix)**

void main()
{
int a = 9;
printf("%d\n", a++);
printf("%d", a);
}
	void main()
	{
	int a = 9;
	printf("%d\n", ++a);
	printf("%d", a);
	}

Output	**Output**
9	10

In above examples, a++ would return the current value of a and then increment the value of a. ++a on the other hand increment the value of a before returning the value. So ++a is called prefix and a++ is called postfix.

Operation	**Operator**	**Example**	**Description**
Pre-increment	++	++a	Increment a by 1; use new value of a in the expression
Post-increment	++	a++	Use current value of a in the expression in which a resides, then increment a by 1
Pre-decrement	--	--b	Decrement b by 1; use new value of b in the expression
Post-decrement	--	b--	Use current value of b in the expression in which b resides, then decrement b by 1

The following table illustrates the difference between the prefix and postfix modes of the increment and decrement operator. Here a=10, r = 10;

Operation	Equivalent Operation	r	a
r = a++;	r = a;a = a + 1	10	11
r = ++a;	a = a + 1;r = a;	11	11
r = a --;	r = a;a = a – 1;	10	9
r = --a;	a = a – 1;r = a;	9	9

2.9.2 Relational Operators

When we want to compare the values of two operands, then relational operators are used. If we want to check that is one operand is equal to or greater than other operands, then we use >= operator.

The below table lists of the relational operators in C with their functions.

Operator	Function	Example
==	This will check if two operands are equal	6 == 2 returns 0
!=	This will check if two operands are not equal.	6 != 2 returns 1
>	This will check if the operand on the left is greater than operand on the right	6 > 2 returns 1
<	This will check if the operand on the left is smaller than the right operand	6 < 2 returns 0
>=	This will check if the left operand is greater than or equal to the right operand	6 >= 2 returns 1
<=	This will check if operand on left is smaller than or equal to the right operand	6 <= 2 return 0

➢ **Example :** C Program using logical operators

```
#include <stdio.h>
int main()
{
int a = 7, b = 7, c = 10;
printf("%d == %d = %d \n", a, b, a == b); // true
printf("%d == %d = %d \n", a, c, a == c); // false
printf("%d > %d = %d \n", a, b, a > b); //false
printf("%d > %d = %d \n", a, c, a > c); //false
printf("%d < %d = %d \n", a, b, a < b); //false
printf("%d < %d = %d \n", a, c, a < c); //true
printf("%d != %d = %d \n", a, b, a != b); //false
```

```
printf("%d != %d = %d \n", a, c, a != c); //true
printf("%d >= %d = %d \n", a, b, a >= b); //true
printf("%d >= %d = %d \n", a, c, a >= c); //false
printf("%d <= %d = %d \n", a, b, a <= b); //true
printf("%d <= %d = %d \n", a, c, a <= c); //true
return 0;
}
```

Output:

```
7 == 7 = 1
7 == 10 = 0
7 > 7 = 0
7 > 10 = 0
7 < 7 = 0
7 < 10 = 1
7 != 7 = 0
7 != 10 = 1
7 >= 7 = 1
7 >= 10 = 0
7 <= 7 = 1
7 <= 10 = 1
```

2.9.3 Logical Operators

Logical Operators are used for True or False results.

The table below lists out the logical operators used in C

Operator	Function	Example (if a=1 and b=0)
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

➢ **Example:** C Program using logical operators.

```
#include <stdio.h>
int main()
{
int a = 8, b = 8, c = 12, result;
result = (a == b) && (c > b);
printf("(a == b) && (c > b) equals to %d \n", result);
result = (a == b) && (c < b);
printf("(a == b) && (c < b) equals to %d \n", result);
```

```
result = (a == b) || (c < b);
printf("(a == b) || (c < b) equals to %d \n", result);
result = (a != b) || (c < b);
printf("(a != b) || (c < b) equals to %d \n", result);
result = !(a != b);
printf("!(a == b) equals to %d \n", result);
result = !(a == b);
printf("!(a == b) equals to %d \n", result);
return 0;
```

Output :

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

2.9.4 Bitwise Operators

These operators are used for bit-level operations on the operands. The operators are converted first to bit-level, and then calculations are performed.

Operator	Function
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

➢ **Example :** C program for Bitwise AND

```
#include <stdio.h>
int main()
{
int a = 10, b = 8;
printf("Output = %d", a&b);
return 0;
}
```

Output = 8

Explanation:

10 = 00001010 (In Binary)

8 = 00001000 (In Binary)

Bit Operation of 10 and 8

00001010 & 00001000 = 00001000 = 8 (In decimal)

2.9.5 Assignment Operators

These types of operators are used to assign a value to a variable.

Operator	Function	Example
=	This will assign values from right side operands to left side operand	a=b
+=	This will add the right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	This will subtract the right operand from the left operand and assign the result to the left operand	a-=b is same as a=a-b
*=	This will multiply the left operand with the right operand and assign the result to the left operand	a*=b is same as a=a*b
/=	This will divide the left operand with the right operand and assign the result to the left operand	a/=b is same as a=a/b
%=	This will calculate modulus using two operands and assign the result to the left operand	a%=b is the same as a=a%b

2.9.6 Conditional Operators

Also known as Ternary Operator or? : Operator. These are used for decision-making.

Syntax:

```
Condition? Expression1: Expression2;
```

Above syntax is equivalent to:

```
if (Condition)
Expression1
else
Expression2
```

Example Program:

Using if/else statement:

```
void main()
{
int total=77;
if (total > 60)
printf("You Passed");
else
printf("You Failed");
}
```

Using conditional statement:

```
void main()
{
int total=77;
printf("You %s", total > 60? "Passed": "Failed");
}
```

Output will be same for both of the above example programs:

```
You Passed
```

2.9.7 Special Operators

Here are some special operators used in C

Operator	Function
&	This operator is used to get the address of the variable. **Example :** &a will give an address of a.
*	This operator is used as a pointer to a variable. **Example :** * a where * is a pointer to the variable a.
size of ()	This operator gives the size of the variable. **Example :** The size of (char) will give us 1.

➢ **Example :** C program using a special operator

```
#include <stdio.h>
int main()
{
int *ptr, q;
q = 40;
/* address of q is assigned to ptr */
ptr = &q;
/* display q's value using ptr variable */
printf("%d", *ptr);
return 0;
}
```

Output :

```
40
```

2.9.8 The Size of Operator

The size of operator returns the size of the operand in bytes used within memory. The operand may be a variable, a constant or a data type qualifier.

The general syntax of Sizeof operator is :

```
sizeof (operand)
```

For Example,

```
#include <stdio.h>
int main(void)
{
char c;
int sum;
long int nationaldebt;
float interest=10.5f;
const char letter='A';
const int age=22;
const float PI=3.14f;
clrscr();
printf("Size of char in memory:%d bytes\n", sizeof (c));
printf("Size of int in memory:%d bytes\n", sizeof (sum));
printf("Size of long int in memory:%d bytes\n", sizeof (nationaldebt));
printf("Size of float in memory:%d bytes\n", sizeof (interest));
printf("Size of char constant in memory:%d bytes\n", sizeof (letter));
printf("Size of int constant in memory:%d bytes\n", sizeof (age));
printf("Size of float constant in memory:%d bytes\n", sizeof (PI));
return 0;
}
```

Output

```
Size of char in memory:1 bytes
Size of int in memory:2 bytes
Size of long int in memory:4 bytes
Size of float in memory:4 bytes
Size of char constant in memory:1 bytes
Size of int constant in memory:2 bytes
Size of float constant in memory:4 bytes
```

The sizeof operator is a compile time operator.

2.10 PRIORITY (PRECEDENCE) AND ASSOCIATIVITY OF OPERATORS

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.Let's understand the precedence by the example given below:st Salary in

```
int value=10+20*10;
```

The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift<<>>	Left to right	
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

The following program illustrates the effect of presence of parenthesis in expressions.

```
void main ()
{
float a, b, c x, y, z;
a = 9;
b = 12;
c = 3;
x = a – b / 3 + c * 2 – 1;
y = a – b / (3 + c) * (2 – 1);
z = a – ( b / (3 + c) * 2) – 1;
printf ("x = %fn",x);
printf ("y = %fn",y);
printf ("z = %fn",z);
}
```

Output of above program:

```
x = 10.00
y = 7.00
z = 4.00
```

C Expressions

An expression is a combination of variables constants and operators in C language. In C every expression evaluates to a value i.e., every expression result in some value of a certain type that can be assigned to a variable

Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form:

```
variable = expression;
```

When the statement is executed, the expression is evaluated first and then result is assigned to variable on the left-hand side. Here are some of the examples of expressions,

```
int a,b,c;
a=b+c;
a=b-c;
a=b/c;
```

Clubbing of operators

We can use more than one operator in expressions. This is called clubbing of operators. Examples of expressions with clubbing of operators are:

```
x =,a * b – c;
y = b / c * a;
z = a – b / c + d;
```

In above statements assuming the value of a=10, b=20, c=5, d=7:

Arithmetic Expression	Operators Clubbed	Possible Arithmetic Operations	Possible sequence of evaluation	Result
a * b – c	* and -	(1) a * b (2) b – c	(1),(2) or (2),(1)	95 $\sqrt{150}$
b / c * a	/ and *	(1) b / c (2) c * a	(1),(2) or (2),(1)	0 $\sqrt{0}$
a – b / c + d	- , / and +	(1) a – b (2) b / c (3) c + d	(1),(2),(3) or (2),(3),(1) or (3),(1),(2) or (2),(1),(3) ...	6 17 9 $13\sqrt{}$

Now in above table our questions: Which is the correct possible sequence of evaluation? The answer to this question is given by Priority (Precedence) and associativity of operators.

2.11 FORMATTED AND UNFORMATTED INPUT AND OUTPUT IN C

Input means to provide the program with some data to be used in the programand Output means to display data on screen or write the data to a printer or a file.C programming language provides many built-in functions to read any given inputand to display data on screen when there is a need to output the result.C programming language has standard libraries that allow input and output in aprogram. The stdio.h or standard input output library in C that has methods for inputand output.

Input output built-in functions in C falls into two categories, namely, formatted inputoutput (I/O) functions and unformatted input output (I/O) functions.printf() and scanf() are examples for formatted input and output functions andgetch(), getche(), getchar(), gets(), puts(), putchar() etc. are examples of unformattedinput output functions.

Difference between Formatted and Unformatted Functions

Formatted I/O functions allow to supply input or display output in user desired format.

Unformatted I/O functions are the most basic form of input and output and they do not allow to supply input or display output in user desired format.

Formatted input and output functions contain format specifier in their syntax.

Unformatted input and output functions do not contain format specifier in their syntax.

Formatted I/O functions are used for storing data more user friendly.

Unformatted I/O functions are used for storing data more compactly.

Formatted I/O Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a;
 clrscr();
 printf("Enter value of a:");
 scanf("%d", &a);
 printf(" a = %d", a);
 getch();s
}
```

Output of the above program is:

```
Enter value of a:5↵
a = 5
```

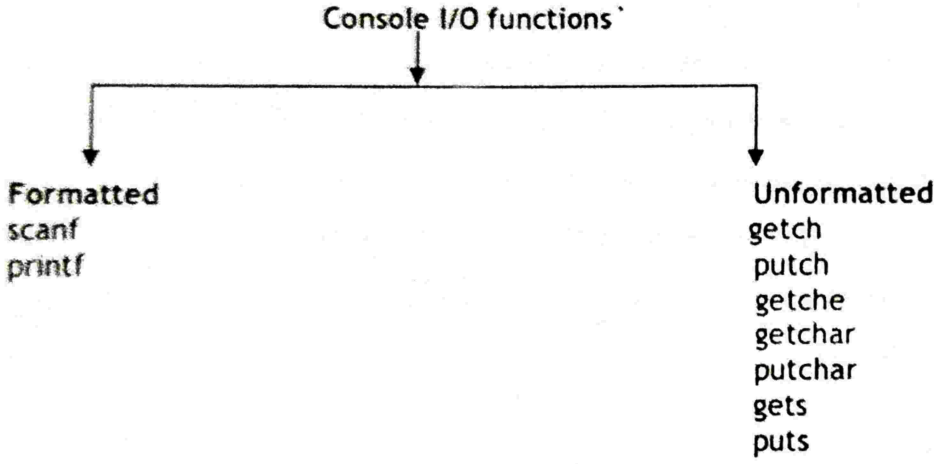
Unformatted I/O Example:

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
 char ch ;
 clrscr();
 printf("Press any character:");
 ch = getche();
 printf("\nYou pressed :"
 putchar(ch);
getch();
}
```

Output of the above program is:

```
Press any character: L
You pressed: L
```



2.11.1 Formatted I/O Functions

Formatting input or output can control how numeric, float or string data in input or output? Formatting Input/ output include:

Size (Width of output)

Position (Left/Right alignment)

Number of digits

Padding

Formatted functions allow us to control where o/p would appear on screen, number of places after decimal point, spaces between two values etc. Let's start with formatted functions:

PRINTF() FUNCTION

This function is used to display output on screen. i.e. it moves data from computer memory to output device. The syntax of the function is:

```
printf ("format string", arg1, arg2...) ;
```

printf() function example:

```
# include<stdio.h>
void main()
{
int avg  =  345 ;
float per  = 69.2f ;
char grade = 'B' ;
printf ("Average =  %d \n Percentage = %f  \n Grade  = %c", avg, per, grade) ;
}
```

Output of above example:

```
Average = 345
Percentage = 69.2
Grade = B
```

The printf function starts examining the format string from left to right..

- **SCANF() FUNCTION**

Data can be entered/input into the computer from an input device (like keyboard) using the scanf function.

The syntax of the function is:

```
scanf ("format string",   &arg1, &arg2 ...) ;
```

scanf() function example:

```
# include<stdio.h>
void main()
{
int   avg ;
float  per ;
char   grade ;
scanf ("%d %f %c" , &avg, &per ,&grade) ;
printf ("Average =  %d \n Percentage = %f  \n Grade  = %c", avg, per, grade) ;
}
```

Output of above example:

```
Average = 345
Percentage = 69.2
Grade = B
```

2.11.2 UnFormatted I/O Functions

The following are the unformatted function used to input and output characters.

- **GETCH() FUNCTION**

getch() function is used to input a single character. It will instantly read the character and does not require enter key to be pressed. It returns the character typed but does not echo (display) it on the screen. The syntax of getch() function is:

```
int getch (void) ;
```

Or

```
ch = getch();
```

This function is generally used at the end of the program to hold output to stay on the screen until user press any key.

getch() function example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
printf("Waiting for key to be pressed to exit.\n");
getch();
return 0;
}
```

Output : No output will be there.

■ GETCHE() FUNCTION

getche() is used to get a character from input device (like keyboard), and echoes (displays) to the screen. The syntax of getche() function is:

```
int getche(void) ;
```

Or

```
ch = getche();
```

getche() function example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
printf("Waiting for key to be pressed to exit.\n");
getche();
return 0;
}
```

Output:

Run this program and press a character. Then view the user screen (Alt+F5) if using turbo c. You will find the character printed on the screen if you pressed a printable character. Try pressing enter or tab key (non printable) characters also.

■ GETCHAR() FUNCTION

It is used to input a single character. getchar() is used to get or read the input (i.e a single character) at run time. The syntax of getchar() function is:

```
int getchar(void);
```

Example Declaration:

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

getchar() function example:

```
#include<stdio.h>
int main()
{
char ch;
ch = getchar();
printf("Input Char Is:%c",ch);
return 0;
}
```

Output (if you press A then, i.e. it depends on key pressed):

```
Input Char Is:A
```

- **PUTCHAR() FUNCTION**

It is other side of getchar. It displays a single character on the screen. The syntax of putchar() function is:

```
putchar(ch) ;
```

putchar() function example:

```
#include<stdio.h>
int main()
{
char ch;
ch = getchar();
putchar(ch);
return 0;
}
```

Output (if you press A then, i.e. it depends on key pressed):

```
Input Char Is:A
```

Above functions are alternative to scanf() and printf() functions used for inputting and outputting characters.

- **GETS() AND PUTS() FUNCTIONS**

gets() and puts() functions facilitate transfer of strings between computer and standard input-output devices. They accept single argument. When using gets, enter key has to be pressed to end the string.

gets() and puts() function example:

```
#include<stdio.h>
int main ( )
{
char line [80] ;
gets (line) ;
puts (line) ;
}
```

Output of above example (if you enter MY WORLD):

```
MY WORLD
```

2.12 FORMATTED INPUT AND OUTPUT IN C

Conversion specifiers tell how the arguments are to be displayed. %d, %f, %c are called conversion specifiers. For example, in following statement:

```
printf ("Average =  %d \n Percentage = %f  \n Grade  = %c", avg, per, grade) ;
```

%d tells indicates avg is to be displayed as an integer,

%f indicates per as float and

%c indicates grade as character.

Field width specifiers can be used with conversion specifiers. Also, whether the value is to be left justified or right justified can be mentioned.

■ FORMATTING OUTPUT

Each specifier can be preceded by a modifier which determines how the value will be printed. The most general modifier is of the form:

```
flag width.precision
```

The flag can be any of:

Flag	Meaning
–	left justify
+	always display sign
space	display space if there is no sign
0	pad with leading zeros
#	use alternate form of specifier

The width specifies the number of characters used in total to display the value and precision indicates the number of characters used after the decimal point.

Specifying Field Width for values

For integers

To display output in some number of spaces on the screen, you can add an integer value after the percent sign of a format specifier. For example, if you want to display an integer using a minimum of 8 spaces, you'd write %8d in your printf() statement. This example gives a demonstration:

```
#include <stdio.h>
int main()
{
  int x = 123;

  printf("Printing 123 using %%0d displays %0d\n", x);
  printf("Printing 123 using %%1d displays %1d\n", x);
  printf("Printing 123 using %%2d displays %2d\n", x);
  printf("Printing 123 using %%3d displays %3d\n", x);
  printf("Printing 123 using %%4d displays %4d\n", x);
  printf("Printing 123 using %%5d displays %5d\n", x);
  printf("Printing 123 using %%6d displays %6d\n", x);
  printf("Printing 123 using %%7d displays %7d\n", x);
  printf("Printing 123 using %%8d displays %8d\n", x);
  printf("Printing 123 using %%9d displays %9d\n", x);

  return 0;
}
```

Output:

```
Printing 123 using %0d displays 123
Printing 123 using %1d displays 123
Printing 123 using %2d displays 123
Printing 123 using %3d displays 123
Printing 123 using %4d displays  123
Printing 123 using %5d displays   123
Printing 123 using %6d displays    123
Printing 123 using %7d displays     123
Printing 123 using %8d displays      123
Printing 123 using %9d displays       123
```

But also, if you write %09d, the program will display zeros before the number itself. In the above example, it will display:

Printing 123 using %09d displays 000000123

EXERCISE

- MCQ

1. Which of the following are tokens in C?

   (a) Keywords (b) Variables (c) Constants (d) All of the above

   Ans. : (d)

2. What is the valid range of numbers for int type of data?

   (a) 0 to 256 (b) -32768 to +32767 (c) -65536 to +65536 (d) No specific range

   Ans. : (b)

3. Which symbol is used as a statement terminator in C?

   (a) ! (b) # (c) ~ (d) ;

   Ans. : (d)

4. Which escape character can be used to begin a new line in C?

   (a) \a (b) \b (c) \m (d) \n

   Ans. : (d)

5. Which escape character can be used to beep from speaker in C?

   (a) \a (b) \b (c) \m (d) \n

   Ans. : (a)

6. Character constants should be enclosed between _____ .

   (a) Single quotes (b) Double quotes (c) Both a and b (d) None of these

   Ans. : (a)

7. String constants should be enclosed between _____ .

   (a) Single quotes (b) Double quotes (c) Both a and b (d) None of these

   Ans. : (b)

8. Which of the following is an example of compounded assignment statement?

   (a) a=5 (b) a+=5 (c) a=b=c (d) a=b

   Ans. : (b)

9. The operator && is an example for _____ operator.

   (a) Assignment (b) Increment (c) Logical (d) Rational

   Ans. : (c)

10. The operator & is used for

    (a) Bitwise AND (b) Bitwise OR (c) Logical AND (d) Logical OR

    Ans. : (a)

11. The operator / can be applied to

    (a) integer values (b) float values (c) double values (d) All of these

    Ans. : (b)

12. The equality operator is represented by

(a) := (b) .EQ. (c) = (d) ==

Ans. : (d)

13. Operators have hierarchy. It is useful to know which operator

(a) is most important (b) is used first (c) is faster (d) operates on large numbers

Ans. : (b)

14. The bitwise AND operator is used for

(a) Masking (b) Comparison (c) Division (d) Shifting bits

Ans. : (a)

15. The bitwise OR operator is used to

(a) set the desired bits to 1 (b) set the desired bits to 0

(c) divide numbers (d) multiply numbers

Ans. : (a)

16. Which of the following operator has the highest precedence?

(a) * (b) == (c) => (d) +

Ans. : (d)

17. In expression i = g() + f(), first function called depends on _____.

(a) Compiler (b) Associativiy of () operator

(c) Precedence of () and + operator (d) Left to write of the expression

Ans. : (a)

18. The associativity of ! operator is

(a) Right to Left (b) Left to Right

(c) (a) for Arithmetic and (b) for Relational (d) (a) for Relational and (b) for Arithmetic

Ans. : (a)

19. Which operator has the lowest priority?

(a) ++ (b) % (c) + (d) ||

Ans. : (d)

20. Operators have precedence. Precedence determines which operator is

(a) faster (b) takes less memory

(c) evaluated first (d) takes no arguments

Ans. : (c)

21. Which of the following is a ternary operator?

(a) ?: (b) * (c) sizeof (d) ^

Ans. : (a)

22. Explicit type conversion is known as

(a) Casting (b) Conversion (c) Disjunction (d) Separation

Ans. : (a)

Questions

1. What is token? Explain different types of tokens in C language.
2. Explain keyword with example.
3. Explain scope and lifetime of variable using example.
4. What is escape sequence? Give examples.
5. Explain different datatype of C language including keyword, memory requirement in bytes, format specifiers, range.
6. Explain primary datatype of C language.
7. Explain User defined datatype of C language.
8. Explain difference between signed and unsigned numbers.
9. What is the purpose of short and long modifiers?
10. What is type conversion?
11. What is implicit type conversion? Explain giving example.
12. What is explicit type conversion? Explain giving example.

Programming Exercise

1. Write a program to display your name on the monitor.
2. Modify the above program to display your address and phone number on separate lines by adding two additional printf() statements.
3. Write a program that writes your name on the monitor ten times.
4. Write a micro to find largest among number.
5. We want to write a program that reads 3 numbers from the keyboard than prints on the screen the largest one. Your program must use only two variables, one variable to read the numbers from the user and one variable for the largest value. Draw the flowchart of that program
6. Write a similar program that displays your name.

GTU Exam Paper Solution

Questions 1 Marks

1. _____ is the valid identifier in C programminglanguage. **Winter 2019**

(A) float (B) Helloworld (C) avg-mark (D) Hello

Ans. : (D)

2. Find the incorrectstatement: **Winter 2019**

(A) Compiler converts a program into machine code.

(B) Compiler gives logical error list in theprogram.

(C) Compiler shows syntax error list in theprogram.

(D) Compiler creates object file of theprogram

3. _____ is not a keyword of C programminglanguage. **Winter 2019**

(A) array (B) else (C) continue (D) void

Ans. : (A)

4. If the equation in program is: ans = 3 + 2 * 3 – 2 / 2 + 3 then calculate the value of ans. **Winter 2019**

(A) 12 (B) 15 (C) 11 (D) 17

Ans. : (C)

Questions 2 Marks

1. Explain a variable and value of a variable with example. **Winter 2019**

Ans. A variable is a name that can change its value in the program. It is label given to a memory location and used to store a data value. Variable names are case sensitive e.g. 'NAME' and 'name' are different variables.

2. Describe the importance of *main*( ) function in C program **Winter 2019**

Ans. In C, program execution starts from the main() function.

Every C program must contain a main() function.

It usually controls program execution by directing the calls to other functions in the program.

The main function may contain any number of statements. These statements are executed sequentially in the order which they are written.

3. Explain pre-increment and post-increment operator with example. **Winter 2019**

Ans. a++ would return the current value of a and then increment the value of a. ++a on the other hand increment the value of a before returning the value. So ++a is called prefix and a++ is called postfix.

Operation	Operator	Example	Description
Pre-increment	++	++a	Increment a by 1; use new value of a in the expression
Post-increment	++	a++	Use current value of a in the expression in which a resides, then increment a by 1

The following table illustrates the difference between the prefix and postfix modes of the increment and decrement operator. Here a=10, r = 10;

Operation	Equivalent Operation	r	a
r = a++;	r = a;a = a + 1	10	11
r = ++a;	a = a + 1;r = a;	11	11
r = a —;	r = a;a = a – 1;	10	9
r = —a;	a = a – 1;r = a;	9	9

4. List out any four header files used in C program with its purpose. **Winter 2019**

5. What is header file? Give list of header file used in 'C' program. **Winter 2018**

Ans. A header file is a file with extension **.h** which contains pre-defined C function declarations.

stdio.h : Input/Output functions

conio.h : Console Input/Output functions

stdlib.h : General utility functions

math.h : Mathematics functions

string.h : String functions

time.h : Date and time functions

6. Write the given expressions in valid C syntax. **Winter 2019**

(a) $D = ((m^2 - n) \div 2xy^3)+4y$ (b) $Temp = ½ \times (bh^2 - 4m)$

Ans. (a) $D = ((m^2 - n) \div 2xy^3)+4y$

Equivalent C Expression in C:

D=((pow(m,2) – n) ÷ 2*x*pow(y,3))+4y

(b) $Temp = ½ \times (bh^2 - 4m)$

Equivalent C Expression in C:

Temp = (1/2) *((b*pow(h,2) –(4*m)))

7. Explain explicit type conversion with example. **Winter 2019**

Ans. Please refer **2.8.5**

8. Explain C tokens. **Winter 2019**

Ans. Please refer **2.6.1**

9. Find output if the value of float x = 12.142872; **Winter 2019**

(a) printf("% .2f", x); (b) printf("%6.3f ",x);

Ans. (a) 12.14 (b) 12.143

10. List out any four bitwise operators with its purpose. **Winter 2019**

Please refer **2.9.4**

11. Why 'C' is called middle level language? **Winter 2018**

Ans. 'C' language provides facilities of both high levels programming as well as low-level programming. C can be used to develop wide variety of applications including low-level programming applications using facilities of using pointers. Hence it can be called as middle level language.

12. Give difference between compiler and interpreter. **Winter 2018**

Compiler	Interpreter
Compiler scans the whole program in one go.	Translates program one statement at a time.
As it scans the code in one go, the errors (if any) are shown at the end together.	Considering it scans code one line at a time, errors are shown line by line.
Main advantage of compilers is its execution time.	Due to interpreters being slow in executing the object code, it is preferred less.
It converts the source code into object code.	It does not convert source code into object code instead it scans it line by line
It does not require source code for later execution.	It requires source code for later execution.
C, C++, C# etc.	Python, Ruby, Perl, SNOBOL, MATLAB, etc.

13. Explain basic structure of 'C' program. **Winter 2018**

Ans. Please refer **2.3**

14. List back slash character constant. **Winter 2018**

Ans.

Esc. Seq	Purpose	Description
\n	New line	Causes a new line to begin.
\b	Backspace	Moves the cursor one position to the left
\f	Form feed	Advances computer stationary to the top of next page
\'	Single quote	Causes a single quote to be printed
\\	Backslash	Causes a back slash to be printed
\"	Double Quote	Causes a double quote to be printed
\t	Tab	The cursor jumps over 8th character with this esc. Seq.
\r	Carriage return	Takes the cursor to the beginning of the line in which it is placed.
\a	Audible alert	Alerts the user by sounding the speaker inside the computer

15. Explain type casting in 'C'. **Winter 2018**

Ans. Please refer **2.8.5**

16. Write rules for naminga variable **Winter 2018**

Ans. Please refer **2.6.5**

17. Give difference between syntax error and run time error

Ans.

Run-Time Error	Syntax Error
This is a type of error that is encountered while the program is running.	This is an error encountered in the syntax of a sequence of characters or the tokens.
It happens due to the occurrence of an illegal operation.	It happens due to the occurrence of violation of grammar rules .
This type of error is detected while the program is still running.	This type of error is detected while compiling the program.
E.g., Array out of bound, Diving by zero, etc.	E.g., Missing curly braces, semicolons, etc.

18. Explain Basic structure of C program **Summer 2019**

Ans. Please refer **2.3**

19. Differentiates equal to (==) and assignment Operator (=) **Summer 2019**

Ans. equal to (==) Operator : Compare two expressions

e.g if(age==18) checks if age is 18

assignmentOperator(=) : Assigns RHS value to LHS.

e.g age =18 stores 18 into age variable

20. Explain the use of comma operator(,). **Summer 2019**

Ans. The comma operator can be used to link related expression together. Comma-linked lists of expressions are evaluated left to right and value of right most expression is the value of the combined expression. For Example:

```
c = (a=11, b=55, a+b);
```

First the value of 11 is assigned to a, then value of 55 is assigned to b. finally the value of a+b (11+55=66) is assigned to c. Comma operator is frequently used in loops to evaluate different parts of loops.

Comma operator used in for loop:

```
for (a=0, b=10; a <=b; a++)
```

Comma operator used in while loop:

```
while (choice=getchar(), choice != 'N')
```

Comma operator can be used to execute more than one expression in sequence. For example when exchanging values of two variables:

```
temp = a, a = b, b = temp;
```

Above statement is equivalent to following three statements:

```
temp=a;
a=b;
b=temp;
```

21. What is the difference between ++a and a++ **Summer 2019**

Ans. :

Example	Description
++a	Increment a by 1; use new value of a in the expression
a++	Use current value of a in the expression in which a resides, then increment a by 1

Questions 3 Marks

1. Write a C program to interchange the value of two integer variables. Input values from keyboard.

Ans. : **Winter 2019**

```
#include<stdio.h>
 int main()
{
int a, b;
printf("Enter a:");
scanf("%d",&a);
printf("Enter b:");
scanf("%d",&b);
printf("Before swap a=%d b=%d",a,b);
a=a+b;//a=30 (10+20)
```

```
b=a-b;//b=10 (30-20)
a=a-b;//a=20 (30-10)
printf("\n\After swap a=%od b=%od",a,b);
return 0;
}
```

2. Write a C program to find the area of a square. Input value of side from keyboard. **Winter 2019**

Ans. :

```
#include<stdio.h>
int main() {
   int side, area;
   printf("\nEnter the Length of Side : ");
   scanf("%od", &side);
   area = side * side;
   printf("\n\Area of Square : %od", area);
   return (0);
}
```

3. Explain storage class of 'C'. **Winter 2018**

Ans. Please refer **5.10**

4. Explain increment and decrement operators with example. **Winter 2018**

Ans. Please refer **2.9.1**

5. Covert following expression into'C': 9xy/p+1-1/3(m+n) **Winter 2018**

Ans. Expression in C=((9*x*y)/p+1)-(1/(3*(m+n)));

6. Explain ternary operator with suitable example. **Winter 2018**

Ans. Please refer **2.9.6**

7. State why following are in valid variable names : **Winter 2018**

(1) 2sum (2) firstname (3) total$ (4) #definePI=3.14

Ans. (1) 2sum : variable can not start with digit

(2) firstname : space not allowed in variable name

(3) total$: it is valid variable name

(4) #definePI=3.14 : its preprocessor directive

8. Write a program to enter days and convertitin to year, month and days. **Winter 2018**

Ans. :

```
#include <stdio.h>
int main() {
    int ndays, y, m, d;
```

```
    printf("Input no. of days: ");

    scanf("%d", &ndays);

    y = (int) ndays/365;

    ndays = ndays-(365*y);

    m = (int)ndays/30;

    d = (int)ndays-(m*30);

    printf(" %d Year(s) \n %d Month(s) \n %d Day(s)", y, m, d);
    return 0;
}
```

9. What are the different data types available in'C'? Explain any three with an example? **Summer 2019**

Ans. Please refer **2.8**

10. What is variable? List the rules for declaring a variable. **Summer 2019**

Ans. Please refer **2.6.5**

Questions 4 Marks

1. Write a program to find N! value. **Winter 2018**

Ans.

```
#include <stdio.h>
#include<conio.h>
void main()
{
    int i, j, Fact=1;
    clrscr();
    printf("Enter Any Number:\n");
    scanf("%d", &j);
    for(i=j; i>=1; —i)
    {
    Fact = Fact * i;
    }
printf("Factorial of Given Number Is= %d", Fact);
getch();
}
```

2. Explain storage class of 'C'. **Winter 2018**

Ans. Please refer **5.10**

3. List out data types with their range of value used in 'C'. **Winter 2018**

Ans. Please refer **2.8**

Range of values used

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

4. Write a program to print only prime number between 1 to N numbers **Winter 2018**

Ans.

```
#include<stdio.h>
int main(){
    int num,i,count,n;
    printf("Enter max range: ");
    scanf("%d",&n);
    for(num = 1;num<=n;num++){
        count = 0;
        for(i=2;i<=num/2;i++){
            if(num%i==0){
                count++;
                break;
            }
        }
        if(count==0 && num!= 1)
            printf("%d ",num);
    }
return 0;
}
```

Sample output:

```
Enter max range: 50
2 3 5 7 11 13
```

5. Write a program to enter a number and print sum of each digit of number. **Winter 2018**

Ans

```
#include<stdio.h>
 int main()
{
int n,sum=0,m;
printf("Enter a number:");
scanf("%d",&n);
while(n>0)
{
m=n%10;
sum=sum+m;
n=n/10;
}
printf("Sum is=%d",sum);
return 0;
}
```

6. Write a program to convert meters into centimeters. **Summer 2019**

Ans.

```
#include<stdio.h>
int main()
{
double meter = 40.8;
double centimeter;
centimeter = 100 * meter;
printf ("Value in Centimeter is: %.2f \n", centimeter);
return 0;
}
```

7. Write a program to convert kilograms into grams **Summer 2019**

Ans. :

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float kg, g;
    printf("Enter weight in Kilogram: ");
    scanf("%f", &kg);
    g = kg*1000;
    printf("Equivalent weight in Gram = %0.2f", g);
    getch();
    return 0;
}
```

8. Explain Increment and Decrement Operator with proper example. **Summer 2019**

Ans.

The Increment and Decrement Operators:

++ (Increment) operator adds one to operand. val++; is the same as val = val + 1;

— (Decrement) operator subtracts one from operand. val—; is the same as val = val - 1;

It is also possible to use ++i and —i instead of i++ and i—.

Operation	Operator	Example	Description
Pre-increment	++	++a	Increment a by 1; use new value of a in the expression
Post-increment	++	a++	Use current value of a in the expression in which a resides, then increment a by 1
Pre-decrement	--	--b	Decrement b by 1; use new value of b in the expression
Post-decrement	--	b--	Use current value of b in the expression in which b resides, then decrement b by 1

Example :

Operation	Equivalent Operation	r	a
r = a++;	r = a;a = a + 1	10	11
r = ++a;	a = a + 1;r = a;	11	11
r = a --;	r = a;a = a – 1;	10	9
r = --a;	a = a – 1;r = a;	9	9

9. List out types of operators in 'C' and explain size of () operator.

Ans. Please refer **2.9**

Questions 5 Marks

1. Write a C program to print numbers from N1 to N2 where N1 and N2 are input from user. **Winter 2019**

Ans.

```
#include<stdio.h>
int main()
{
    int n1, n2;
    printf ("Enter n1:");
    scanf("%d",&n1);
    printf ("Enter n2:");
    scanf("%d",&n2);

 for (int i=n1;i<n2;i++)
 {
    printf ("%d\n",i);
 }
    return 0;
}
```

Output:

```
Enter n1:5
Enter n2:15
5
6
7
8
```

```
9
10
11
12
13
14
```

Questions 7 Marks

1. Write a menu driven C program that asks user to select an option and perform following task accordingly.

If useropts 1 Find the given number is odd oreven **Winter 2019**

2. Find the factorials of a givennumber
3. Exit theprogram

Print "Invalid Input" message if user opts any other option.

Ans.

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int c=0, num, res, n, flag=0, i;
   while(c!=4)
   {
      //display menu
      printf("\n1    Factorial of a number\n2   Prime or not\n3.  Odd or even\n4 Exit\n");
      //display choice option to the user
      printf("\nEnter your choice:");
      scanf("%d", &c);
      //write case statement for Four options
      switch(c)
      {
         //For factorial block
         case 1:
            //code for factorial functionality
            printf("Enter an integer: ");
            scanf("%d", &num);
            n=num;
            res=num;
            while(num>1)
            {
               res = res*(num-1);
               num = num-1;
            }
            printf("\nFactorial of %d is %d. \n\n",n, res);
            break;
         //For prime block
```

```
        case 2:
            //functionality of Prime or not
            printf("Enter an integer: ");
            scanf("%d", &num);
            n=num;
            for(i=2;i<=n/2;i++)
            {
                if(num%i==0)
                {
                    flag=1;
                    break;
                }
            }
            //for number "1" it's neither prime nor composite
            if(num==1)
                printf("\n1 is neither prime nor composite");
            else
            {
                    if(flag==0)
                        printf("\n%d is Prime Number.\n\n", n);
                    else
                        printf("\n%d is not a Prime Number.\n\n", n);
            }
            break;
        //For Odd-even block
        case 3:
            //functionality for Odd-even
            printf("Enter an integer: ");
            scanf("%d", &num);
            n=num;
            if(num%2==0)
                printf("\n%d is Even Number.\n\n",n);
            else
                printf("\n%d is Odd Number.\n\n",n);
            break;
        //For Exit block
        case 4:
            printf("\nExit");
            break;
    }
  }
}
```

2. Explain working of arithmetic, relational, conditional and sizeof operators with example. **Winter 2019**

Ans. Please refer **2.9**

3. Explain following terms : (1) Dynamic initialization (2) Keywords (3) Identifiers

Ans. Please refer **2.7, 2.6.2, 2.6.3** **Winter 2018**

* * *