

Unit 2

Introduction to C

Assignment

- History of C language
- Features of C
- The C as a middle-level language
- The C as a system programming language

Character set of C

Character set is a package of valid characters recognized by compiler/interpreter.

C uses character as building blocks to form basic program elements.(Eg. Constants, variables, expressions etc.

The characters available in C are categorized into following types.

1) Letters/Alphabets

- Uppercase(A-Z)
- Lowercase(a-z)

2) Digits (0-9)

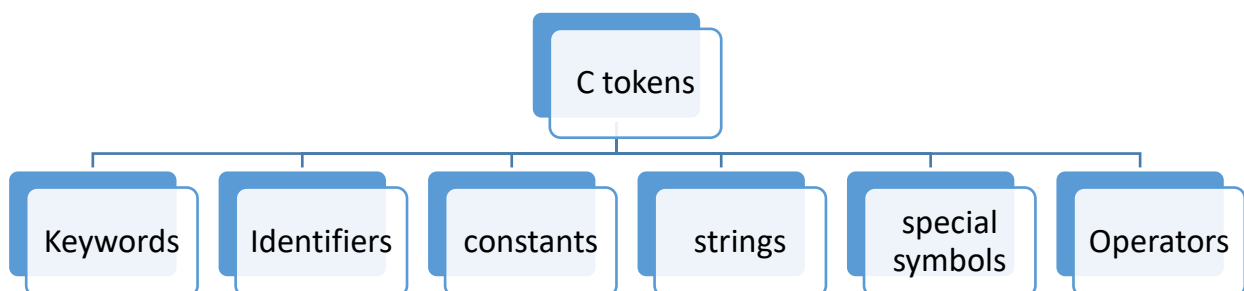
3) Special symbols (+, -, *, !, %, _, <, >, :)

4) Whitespace characters

- Blank Space
- Newline
- Horizontal tab
- Vertical tab

Tokens

Tokens are the smallest individual units of program. C has six types of tokens .They are as follows:



Keywords

Keywords are predefined words whose meaning has already defined to c compilers. These keywords are used for pre-defined purposes and cannot be used as identifier.

The standard keywords are:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	signed	void	for
default	goto	sizeof	volatile
do	if	static	while

Identifiers

Identifiers are the names given by user to various program elements such as variables, function and arrays.

What are the rules for naming identifier? [PU: 2014 spring]

Rules of naming Identifiers

- First character must be alphabet (or underscore).
- Must consists of only letters, digits or underscore
- Keyword cannot be used.
- The length of identifiers should not be greater than 31 characters.
- Must not contain whitespace.
- Identifiers are case-sensitive.

Data type

Important Questions from this topic:

- *What are the different data types available in C? Explain their types qualifier, conversion character, range of value and storage size in memory occupied by each type. [PU:2018 fall]*
- *Mention the appropriate data type for storing following data. Also justify your answer in brief. [PU:2013 fall]*
 - *Distance jumped by frog*
 - *A prime number between 5 and 555*
 - *Weight of your body*
 - *The examination symbol number for student*

- How can you declare following variables using suitable data types? Mobile phone numbers, address, body, temperature, salary. Also explain each memory occupancy size and range. **[PU:2018 spring]**
- Why we need different data types in programming? **[PU:2017 spring]**
- Why it is necessary to have knowledge of data type in C programming. Explain all types of data types of data type available in C. **[PU: 2017 spring]**
- Describe the four basic data-types along with their size and range. **[PU:2019 fall]**

Definition:

Data type is a classification of type of data that a variable can hold in programming. The data type specifies the range of values that can be used and size of memory needed for that value. Generally data type can be categorized as:

1) Primary data type

The size and range of data types on 16 bit machine are:

Data type	Description	Required Memory	Range	Format specifier
char	Used to store character value Eg. 'a', 'c', 'x'	1 byte	-128 to 127	%c
int	Used to store whole numbers/non fractional numbers. Eg. 101, 205, -908	2 bytes	-32768 to +32767	%d
float	Used to store fractional number and has precision of 6 digits. Eg. 5.674, 304.67, 67.8903	4 bytes	-3.4×10^{38} to 3.4×10^{38}	%f
double	Used when precision of float is insufficient and it has precision of 14 digits.	8 bytes	-1.7×10^{308} to 1.7×10^{308}	%lf
void	Has no values and used as return type for function that do not return a value. Eg. void main()			

2) Derived data type

They are derived from existing primary data types.

Eg. Array, Union, Structure

3) User defined data type

The data type that are defined by user is known as user defined data types. Examples **enum** (Enumerated data type) and **typedef** (type definition datatype)

General form of typedef

typedef existing_data_type new_name_for_existing_data_type ;

fundamental data type

new identifier

Example

typedef int integer ;
integer symbolizes int data type Now, we can declare int variable “a” as **integer a** rather than **int a** .

Variable

Variable is defined as a meaningful name given to the data storage location in computer memory. It is a medium to store and manipulate data. The value of variable can change during execution of program.

Variable declaration

Naming a variable along with its data type for programming is known as variable declaration.

The declaration deals with two things:

- It tells the compiler what the variable name is
- It specifies what type of data the variable can hold

Syntax:

data_type variable_name ;

eg. int roll;

roll is a variable of type integer. roll can only store integer variable only.

- Explain the necessary rules to define the variable name in C programming. [PU:2012 fall]
- Which of the following are invalid variable name and why? [PU:2019 fall]

Minimum	First.name	Row total	&name
Doubles	3 rd _row	column_total	float

Rules for declaring variables

Variable name must begin with a letter, some system permit underscore as first character.

- ANSI standard recognizes a length of 31 characters.
- It should not be keyword.
- Whitespace is not allowed.
- Variable name are case sensitive .ie. uppercase and lowercase are different. Thus variable temp is not same as TEMP.
- No two variables of the same name are allowed to be declared in the same scope.

Variable initialization

Initializing variable means specifying an initial value to assign it.

1) Compile time initialization

In compile time initialization value is assigned to variable using assignment operator '='.

Eg: `int a;` //variable declaration
`a=10;` //variable initialization

It is also possible to assign a value at the time of declaration of variable.

Eg. `int a=10,b=5;`
`Float x=10.5;`

2) Runtime initialization

In runtime initialization the value is assigned by using `scanf()` function.

Syntax:

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

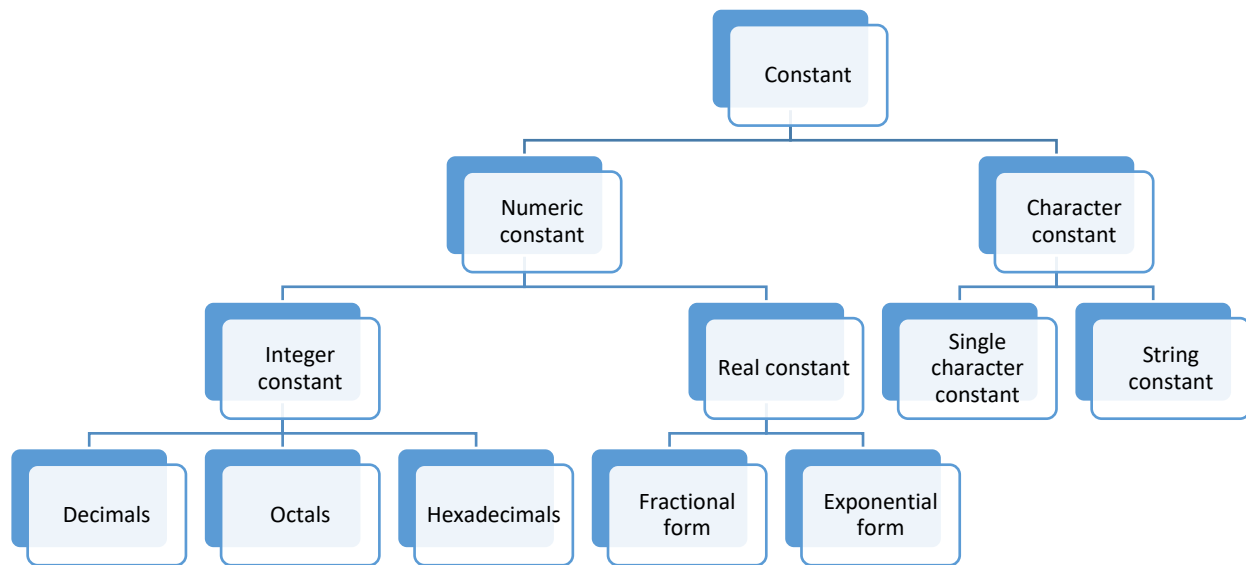
eg. `scanf("%d%d",&a,&b);`

Program to illustrate runtime initialization:

```
#include<stdio.h>
int main()
{
int a,b;
printf("Enter two numbers\n");
scanf("%d%d",&a,&b);
printf("Entered numbers are %d and %d",a,b);
return 0;
}
```

Constants

Constant refers to fixed values that do not change during the execution of program. These fixed values are also called literals. The C constants are as shown in figure. Number associated constant are numeric constant and character associated constant are character constants.



1) Numeric Constant

It consist of:

1.1 Integer constant

It refers to sequence of digits.

There are three types of integer constants.

Decimals: Decimal Integer consist of set of digits, 0 through 9, preceded by an optional +ve or –ve sign.eg.124,-321,678, 654343 etc.

Octal: An octal Integer constant consist of any combination of digits from the set 0 to 7, with a leading 0. eg 037, 075, 0664, 0551

Hexadecimal: Hexadecimal are sequence of digits 0-9 and alphabet A-F with preceding 0x or 0X. eg. 0x5567, 0X53A, 0xFF etc.

1.2. Real or floating point constant

They are numeric constant with decimal points. The real constant are further categorized as

Fractional real constant

They are the set of digits from 0 to 9 with decimal points.eg.394.7867,-0.78676

Exponential real constant

In the exponential form the constants are represented in following form:

mantissa e exponent

where, the mantissa is either a real number expressed in decimal notation or an integer but exponent is always in integer form.

eg.

21565.32=2.156532E4, Where E4=10⁴

Similarly, -0.000000368 is equivalent to -3.68E-7

2) Character Constant

Single Character Constant

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Eg.

'5' 'X' 'A' '4' etc.

String constant

A string constant is a sequence of characters enclosed in double quotes. The characters may be numbers, special characters and blank space.eg. "Hello" "green" "305" etc.

Type casting

Converting an expression of a given type into another type is known as type casting.

Here, it is best practice to convert lower data type to higher data type to avoid data loss.

1. Implicit Conversion

Implicit conversions do not require any operator for conversion. They are automatically performed when a value is copied to a compatible data type in the program.

Here, the value of **i** has been promoted from **int** to **float** and we have not had to specify any type-casting operator

.Program

```
#include<stdio.h>
int main()
{
    int i=20;
    float result;
    result=i; // implicit conversion
    printf("value=%f", result);
    return 0;
}
```

output

Result=20.000000

2) Explicit conversion

In this type of conversion one data type is converted forcefully to another data type by the user.

The general rule for cast is

(type-name) expression

Where type-name is one of the standard c data types. The expression may be constant , variable or an expression.

Example	Action
<code>x=(int)7.5</code>	7.5 is converted to integer by truncation
<code>a=(int)21.3/(int)4.5</code>	Evaluated as 21/4 and result would be 5
<code>b=(double)sum/n</code>	Division is done in floating point mode
<code>z=(int)a+b</code>	a is converted into integer and then added to b

Program

```
#include<stdio.h>
int main()
{
    int a=5,b=2;
    float result;
    result=(float)a/b; // Explicit conversion
    printf("value=%f",result);
    return 0;
}
```

Output

```
Result=2.50000
```


Write a short notes on: Escape sequences [PU: 2016 fall]

The commonly used Escape sequences are as follows

Escape Sequence	Purpose
\a	Alert sound .A beep is generated by a computer on execution.
\b	Backspace
\n	Newline
\r	carriage return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\"	Display double quote character
\'	Display single quote character
\0	Null character. Marks the end of string

Symbolic constant is simply a identifier used in place of constants. They are usually defined as the start of the program. When the program is compiled each occurrence of symbolic constant is replaced by corresponding character sequence. Preprocessor directive like `#define` allow an identifier to have constant value throughout the program.

```
#define symbolic constant name value
```

Advantages:

- Modifiability
- Understandability

Program to illustrate use of symbolic constant

```
#include<stdio.h>
#define PI 3.1416
int main()
{
    float r,area;
    printf("\nEnter the value of r\n");
    scanf("%f",&r);
    area=PI*r*r;
    printf("Area=%f",area);
    return 0;
}
```

ASCII

ASCII is the American Standard Code for Information Interchange. It defines an encoding standard for the set of characters and unique code for each character in the ASCII chart.

A character variable holds ASCII value (an integer number between 0 and 127) rather than that character itself in c programming. That value is known as ASCII value.

for example ASCII value of 'A' is 65

what this means is that, if you assign 'A' to a character variable,65 is stored in that variable rather than 'A' itself.

Characters	ASCII values
A-Z	65-90
a-z	97-122
0-9	48-57

Program to print ASCII value of the given character

```
#include<stdio.h>
int main()
{
    char ch;
    printf("Enter the character\n");
    scanf("%c",&ch);
    printf("ASCII value of %c is %d",ch,ch);
    return 0;
}
```

Input/ Output functions

C programming language provides many built-in functions to read any given input and to display data on screen when there is need to output the result. The header file for input/output functions is <stdio.h>

The input/output function is classified into two types.

- 1) Formatted I/O functions
- 2) Unformatted I/O functions

1) Formatted I/O functions

Formatted Input

Formatted Input refers to an input data that can be arranged in a particular format according to user requirements. The built-in function scanf() can be used to input data into the computer from standard Input device.

The general form of scanf is

scanf("control string",arg1,arg2argn);

-
- The control string refers to field in which data is to be entered.
- The arguments arg1,arg2 ,...argn specify the address of locations where data is stored.

eg. scanf("%d%d",&num1,&num2);

Formatted output

Formatted output functions are used to display data in a particular specified format.

The printf() is an example of formatted output function. The general form of printf() statement is **printf("control string",arg1,arg2 argn);**

The control string may consist of the following data items.

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item.
3. Escape sequence characters such as \n,\t and \b

The arguments arg1,arg2,.....argn are the variables whose value are formatted and printed according to specifications of the control string.

Eg. printf("First number=%d\tSecond number=%d",num1,num2);

Program to read multiple values of different data types using single scanf() function.

```
#include<stdio.h>
int main()
{
    char name[20];
    int age;
    char gender;
    float weight;
    printf("Enter your Name, Age, Gender and Weight\n");
    scanf("%s %d %c %f", name, &age, &gender, &weight);
    printf("Your Name=%s\n", name);
    printf("Your Age=%d\n", age);
    printf("Your Gender=%c\n", gender);
    printf("Your Weight=%0.2f", weight);
    return 0;
}
```

Format specifier

Format specifier is used during input and output. It is the way to tell the compiler what type of data is in variable during taking input using scanf() or printing using printf(). In format specifier the percentage(%) is followed by conversion character. This indicates the corresponding data item.

Example:

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

Here %d is format specifier and it tells the compiler that we want to print an integer value that is present in variable a.

In this way there are several format specifiers in c. Like %c for character, %f for float, etc.

List of different format specifiers

Format specifier	Purpose
%c	Character
%d	Signed Integer
%f	Floating point
%u	Unsigned Integer
%o	Octal
%X or %x	Hexadecimal representation of unsigned integer
%e or %E	Scientific Notation of float values
%s	String

%lf	Floating point (double)
%%	Prints % character
%n	Prints nothing

2) Unformatted I/O Functions

Unformatted I/O function do not allow the user to read or display data in a desired format. These type of library functions basically deal with a single character or string of characters. The functions `getchar()`, `putchar()`, `gets()`, `puts()`, `getch()`, `getche()`, `putch()` are considered as unformatted functions.

i) `getchar()` and `putchar()`

The `getchar()` reads a character from a standard input device. It buffers the input until the 'ENTER' key is pressed and then assigns this character to character variable.

Syntax is :

`character_variable=getchar();`

where `character_variable` refers to some previously declared character variable.

Eg.

`char c;`

`c=getchar();`

`putchar()` function displays a character to standard output device.

Syntax is: `putchar(character_variable);`

Program

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

ii) `getch()`, `getche()` and `putch()`

The function `getch()` and `getche()` both reads a single character in a instant. It is typed without pressing the ENTER key. The difference between them is that `getch()` reads a character typed without echoing it on a screen, while `getche()` reads the character and echoes (displays) it onto the screen.

Syntax:

```
character_variable=getch();  
characher_varibale=getche();
```

The function putch() prints a character onto the screen.

Syntax:

```
putch(character_variable);
```

Program

```
#include<stdio.h>  
#include<conio.h>  
int main()  
{  
char ch1,ch2;  
printf("Enter First character:");  
ch1=getch();  
printf("\nEnter Second character:");  
ch2=getche();  
printf("\n First character is:");  
putch(ch1);  
printf("\nSecond character is:");  
putch(ch2);  
getch();  
return 0;  
}
```

Output

```
Enter First character:  
Enter Second character: b  
First character is: a  
Second character is: b
```

At first input getch() function read the character input from the user but that the entered character was not displayed. But in second input getche() function read the a character from the user input and entered character was echoed to the user without pressing any key.

iii) gets() and puts()

The gets() function is used to read a string of text containing whitespaces, until newline character is encountered. It can be used as an alternative function for reading strings. Unlike scanf() functions, it does not skip whitespaces(ie.It can be used to read multiwords string)

Syntax: gets(string_variable);

The puts() function is used to display the string on the screen.

Syntax: puts(string_variable);

Program

```
#include<stdio.h>
int main()
{
    char name[20];
    printf("Enter your name:\n");
    gets(name);
    printf("Your name is:");
    puts(name);
    getch();
    return 0;
}
```

OPERATOR

Important questions from this topic

1. Define operator in C. List out the different types of operators used in C. Explain three of them with example. **[PU:2015 fall]**
2. Define operator and operand. List the types of operators and explain any five of them. **[PU: 2019 spring]**
3. What is an operator? Explain the arithmetic, logical, relational and assignment operators in C language. **[PU:2016 fall]**
4. What is an operator? Explain conditional operator with suitable example. **[PU:2017 fall]**
5. Describe about the unary operator, binary operator and ternary operator with example. **[PU:2016 spring]**
6. Write a short notes on:
 - Unary operations
 - Binary and unary operators **[PU:2015 fall][PU:2014 spring]**

Operator

- Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in program to manipulate data and variables.

Operand

The data items on which operators are act upon are called operands.

eg. $a + b$

- Here, symbol + is known as operator
- a and b are operands

Expression

The combination of variables, constants and operators written according to syntax of programming language is known as Expression.

eg. $a+b*c-5$

Types of operators

Types of Operators are classified as:

1. On the basis of no of operands
2. On the basis of function/utility

1. On the basis of number of operands

i) Unary operator

The operator which operates on single operand known as unary operator.

(++) increment Operator
(--) Decrement operator
+ Unary plus
- Unary minus
eg . ++x, -5, +8 , y- - etc.

ii) Binary operator

The operator which operates on two operands are known as binary operator.

eg. +(addition), -(subtraction) , /(division) , * (multiplication) , <(less than) >(greater than) etc.

eg 3+3, 5>2 , 6*7 , 15-4

iii) Ternary operators

The Operators that operates on three operands are known as ternary operator.

Ternary operator pair “?:” is available in c .

Syntax:

expression1? expression2:expression3

The expression1 is evaluated first,

- if it is true then expression2 is evaluated and its value becomes value of the expression
- If it is false expression3 is evaluated and its value becomes value of the expression.

Eg .Let us consider the program statement

```
int a=15,b=10,max;
```

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

Here, The value of a will assigned to max.

2. On the basics of utility /functions

1) Arithmetic Operators

They are used to perform Arithmetic/Mathematical operations on operands.

Operator	Meaning	Example
		If a=20 and b=10
+	Addition	a+b=30
-	Subtraction	a-b=10
*	Multiplication	a*b=200
/	Division	a/b=2
%	Modulo Division	a%b=0

2) Relational Operators

These are used to compare two quantities and depending on their relation take certain decision. The value of relational operator is either 1(if the condition is true) and 0 (if the condition is false).

Operator	Meaning	Example
		If a=20 and b=10
==	Equal to	(a==b) is not True
!=	Not Equal to	(a!=b) is True
>	Greater than	(a>b) is True
<	Less than	(a<b) is not True
>=	Greater than or Equal to	(a>=b) is True
<=	Less than or Equal to	(a<=b) is not True

3) Logical Operators

Logical Operators are used to compare or evaluate logical and relational expressions and returns either 0 or 1 depending upon whether expressions results true or false.

Operator	Meaning	Example
If a=20 and b=10		
&&	Logical AND	Expression ((a==b)&&(a>15)) equals to 0
	Logical OR	Expression ((a==b) (a>15)) equals to 1
!	Logical NOT	Expression !(a==b) equals to 1

4) Assignment Operators

Assignment Operators are used to assign the result of an expression to a variable. The mostly used assignment operator is “=”.

Operator	Name	Example
=	Assignment	c=a+b will assign value of a+b into c
+=	Add AND assignment	c+=a is equivalent to c=c+a
-=	Subtract AND assignment	c-=a is equivalent to c=c-a
=	Multiply AND assignment	c=a is equivalent to c=c*a
/=	Divide AND assignment	c/=a is equivalent to c=c/a
%=	Modulus AND assignment	c%=a is equivalent to c=c%a

5) Increment and Decrement Operator

The increment operators (++) causes its operand to be increased by 1 whereas decrement operator (--) causes its operand to be decreased by 1.

They are unary operators (ie. They operate on single operand).It can be written as

- x++ or ++x (post and pre increment)
- x-- or --x (post and pre decrement)

Note: The increment and decrement operators can be used as postfix and prefix notations.

Prefix notation: The variable is incremented (or decremented) first and then expression is evaluated using the new value of the variable.

Eg. int m=5;
 y=++m;
 In this case the value of y and m would be 6.

Postfix notation: The expression is evaluated first using the original value of the variable and then variable is incremented (or decremented by one).

Eg. int m=5;
 y=m++;

The value of y would be 5 and m would be 6

6) Conditional Operator

The ternary Operator pair “?:” is available in c to construct conditional expressions of the form.

expression1? expression2: expression3

The expression1 is evaluated first,

- if it is true then the expression 2 is evaluated and its value becomes the value of the expression.
- If expression1 is false expression3 is evaluated and its value becomes the value of the expression.

Example:

Consider the following statements.

```
int a, b;  
a=10;  
b=15;  
max= (a>b)? a: b;  
In this example value of b will be assigned to max.
```

Program to find the maximum number between two numbers using conditional operator.

```
#include<stdio.h>  
int main( )  
{  
    int num1,num2,max;  
    printf("\nEnter the two numbers:\n");  
    scanf("%d%d",&num1,&num2);  
    max=(num1>num2)?num1:num2;  
    printf("\nThe largest number is %d",max);  
    return 0;  
}
```

7) Bitwise operators

The operators which are used for the manipulation of data at bit level. These operators are used for testing the bits, shifting them right or left.

Operator	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

The truth tables for &, |, and ^ is as follows:

P	Q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Assume a = 60 and b = 13 in binary format, they will be as follows –

```
a = 0011 1100
b = 0000 1101
-----
a&b = 0000 1100    =>12
a|b = 0011 1101    =>61
a^b = 0011 0001    => 49
```

Bitwise shift operators

i. Left shift <<

This causes the operand to be shifted left by some bit positions.

General form:

Operand<<n

Eg.

n=60, n1=n<<3

N	0000 0000	0011 1100
First shift	0000 0000	0111 1000
Second shift	0000 0000	1111 0000
Third shift	0000 0001	1110 0000

After left shifting value of n1 becomes 480

ii. Right shift >>

This causes operand to be shifted to the right by some bit positions.

Operand >>n

Eg. n=60, n2=n>>3

N	0000 0000 0011 1100
First shift	0000 0000 0001 1110
Second shift	0000 0000 0000 1111
Third shift	0000 0000 0000 0111

After Right shifting value of n2 becomes 7

8) Special operators

C supports some special operators. Some of them are

i. Sizeof operator

The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may be variable, constant or data type qualifier.

Examples:

```
m=sizeof(sum);
n=sizeof(long int);
k=sizeof(235L);
```

Program

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float num;
    printf("Number of bytes allocated =%d",sizeof(num));
    getch();
    return 0;
}
```

ii. Comma operator

The comma operator can be used to link related expressions together. A comma-linked list are evaluated left to right and the value of right most expression is the value of combined expression.

For example the statement

```
value = (x=10, y=5, x+y);
```

First assigns the value 10 to x then assigns 5 to y and finally assigns 15 (ie 10+5) to value.

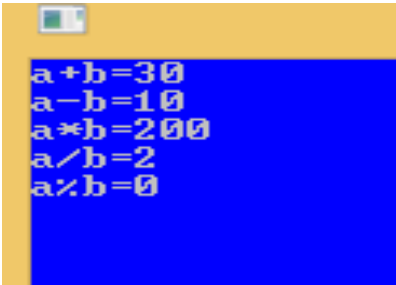
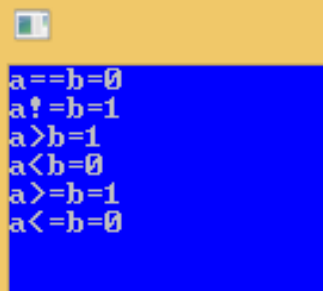
Operator precedence and associativity

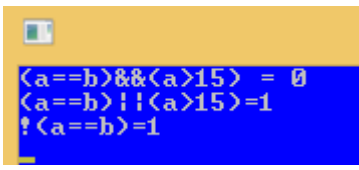

Write a short notes on: Operator precedence and associativity [PU:2018 fall]


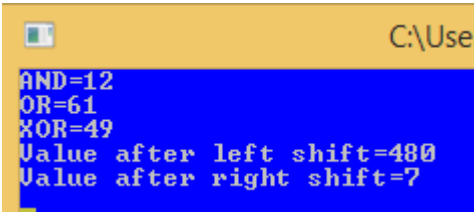
- Operator precedence is a predefined rule of priority of operators. If more than one operators are involved in an expression the operator of higher precedence is evaluated first.
- Associativity indicates the order in which multiple operators of same precedence executes.

Precedence	Operator	Associativity
1	(), {}	Left to right
2	++, --, !	Right to left
3	*, /, %	Left to right
4	+, -	Left to right
5	<, <=, >, >=	Left to right
6	==, !=	Left to right
7	&&	Left to right
8		Left to right
9	?:	Right to left
10	=, *=, /=, %=, -=	Right to left

Some operators Examples:

Program to illustrate Arithmetic Operators	Program to illustrate Relational Operators
<pre> 1 #include<stdio.h> 2 #include<conio.h> 3 int main() 4 { 5 6 int a=20, b=10, c; 7 c=a+b; 8 printf("a+b=%d\n", c); 9 c=a-b; 10 printf("a-b=%d\n", c); 11 c=a*b; 12 printf("a*b=%d\n", c); 13 c=a/b; 14 printf("a/b=%d\n", c); 15 c=a%b; 16 printf("a%%b=%d", c); 17 getch(); 18 return 0; 19 }</pre>	<pre> 1 #include<stdio.h> 2 #include<conio.h> 3 int main() 4 { 5 int a=20, b=10; 6 printf("a==b=%d\n", a==b); 7 printf("a!=b=%d\n", a!=b); 8 printf("a>b=%d\n", a>b); 9 printf("a<b=%d\n", a<b); 10 printf("a>=b=%d\n", a>=b); 11 printf("a<=b=%d\n", a<=b); 12 getch(); 13 return 0; 14 }</pre>
<p>Output</p> 	<p>Output</p> 

Program to illustrate logical operators	Program to illustrate assignment operators
<pre> 1 #include<stdio.h> 2 #include<conio.h> 3 int main() 4 { 5 int a=20, b=15, result; 6 result=(a==b) &&(a>15); 7 printf(" (a==b) &&(a>15) = %d\n", result); 8 result=(a==b) (a>15); 9 printf(" (a==b) (a>15)=%d\n", result); 10 result=! (a==b); 11 printf(" ! (a==b)=%d\n", result); 12 getch(); 13 return 0; 14 } </pre>	<pre> 1 #include<stdio.h> 2 #include<conio.h> 3 int main() 4 { 5 int a=5, c; 6 c=a; 7 printf(" c=%d\n", c); 8 c+=a; 9 printf(" c=%d\n", c); 10 c-=a; 11 printf(" c=%d\n", c); 12 c*=a; 13 printf(" c=%d\n", c); 14 c/=a; 15 printf(" c=%d\n", c); 16 c%=a; 17 printf(" c=%d\n", c); 18 getch(); 19 return 0; 20 } 21 </pre>
<p>Output</p> 	<p>output</p> 

Program to illustrate Increment/Decrement Operators	Program to illustrate Bitwise operators
<pre> 1 #include<stdio.h> 2 #include<conio.h> 3 int main() 4 { 5 int a=5, b=10, c=15, d=20, result; 6 result=++a; 7 printf("%d\n", result); 8 result=b++; 9 printf("%d\n", result); 10 result=--c; 11 printf("%d\n", result); 12 result=d--; 13 printf("%d\n", result); 14 getch(); 15 return 0; 16 } 17 </pre>	<pre> #include<stdio.h> #include<conio.h> int main() { int n1=60,n2=13,x,y,z,left,right; x=n1&n2; y=n1 n2; z=n1^n2; left=n1<<3; right=n1>>3; printf("AND=%d\n",x); printf("OR=%d\n",y); printf("XOR=%d\n",z); printf("Value after left shift=%d\n",left); printf("Value after right shift=%d\n",right); getch(); return 0; } </pre>
<p>Output</p> 	<p>output</p> 

1) WAP to display your name.

```
#include<stdio.h>
int main()
{
char name[20];
printf("Enter your name:");
gets(name);
printf("\nYour name is:");
puts(name);
return 0;
}
```

2) WAP to calculate area and circumference of circle having the radius r should be taken from user.

```
#include<stdio.h>
int main()
{
float r,area,circum;
printf("Enter the radius of circle");
scanf("%f",&r);
area=3.1416*r*r;
circum=2*3.1416*r;
printf("\nArea of circle=%f",area);
printf("\nCircumference of circle is %f",circum);
return 0;
}
```

3) WAP to swap two numbers

```
#include<stdio.h>
int main()
{
int a,b,temp;
a=5;
b=10;
printf("Number before swapping a=%d
and b=%d",a,b);
temp=a;
a=b;
b=temp;
printf("\nNumber after swapping a=%d
and b=%d",a,b);
return 0;
}
```

4) WAP to swap two numbers without using third variable.

```
#include<stdio.h>
int main()
{
int a,b;
a=5;
b=10;
printf("\nNumber before swappig a=%d and
b=%d",a,b);
a=b+a;
b=a-b;
a=a-b;
printf("\nNumber after swappig a=%d and b=%d",a,b);
return 0;
}
```

<p>5) WAP that will convert temperature in centigrade into Fahrenheit.</p> <pre> #include<stdio.h> int main() { float centi,fah; printf("\nEnter the temperature in centigrade"); scanf("%f",&centi); fah=1.8*centi+32; printf("\nThe equivalent temperature in fahrenheit is %f",fah); return 0; } </pre>	<p>6) WAP to check the given 3 digit number is armstrong or not.</p> <pre> #include<stdio.h> #include<math.h> int main() { int num,a,b,c,arm; printf("\nEnter the number"); scanf("%d",&num); c=num%10; b=(num/10)%10; a=num/100; arm=pow(a,3)+pow(b,3)+pow(c,3); if(num==arm) { printf("Entered number is armstrong number"); } else { printf("Entered number is not armstrong number"); } return 0; } </pre>
<p>7) WAP to find the reverse of a given (3 digit) number.</p> <pre> #include<stdio.h> int main() { int num,a,b,c,rev; printf("Enter the number"); scanf("%d",&num); c=num%10; b=(num/10)%10; a=num/100; rev=c*100+b*10+a; printf("Reverse of given number is %d",rev); return 0; } </pre>	<p>8) WAP to find the sum of digit of given (3 digit) number.</p> <pre> #include<stdio.h> int main() { int num,a,b,c,sum; printf("Enter the number"); scanf("%d",&num); c=num%10; b=(num/10)%10; a=num/100; sum=a+b+c; printf("Sum of digit of given number is %d",sum); return 0; } </pre>

9) WAP to check the given year is leap year or not.

```
#include<stdio.h>
int main()
{
    int year;
    printf("Enter the year");
    scanf("%d",&year);
    if((year%4==0&&year%100!=0) || year%400==0)
    {
        printf("%d is leap year",year);
    }
    else
    {
        printf("%d is not leap year",year);
    }
    return 0;
}
```