Module 2

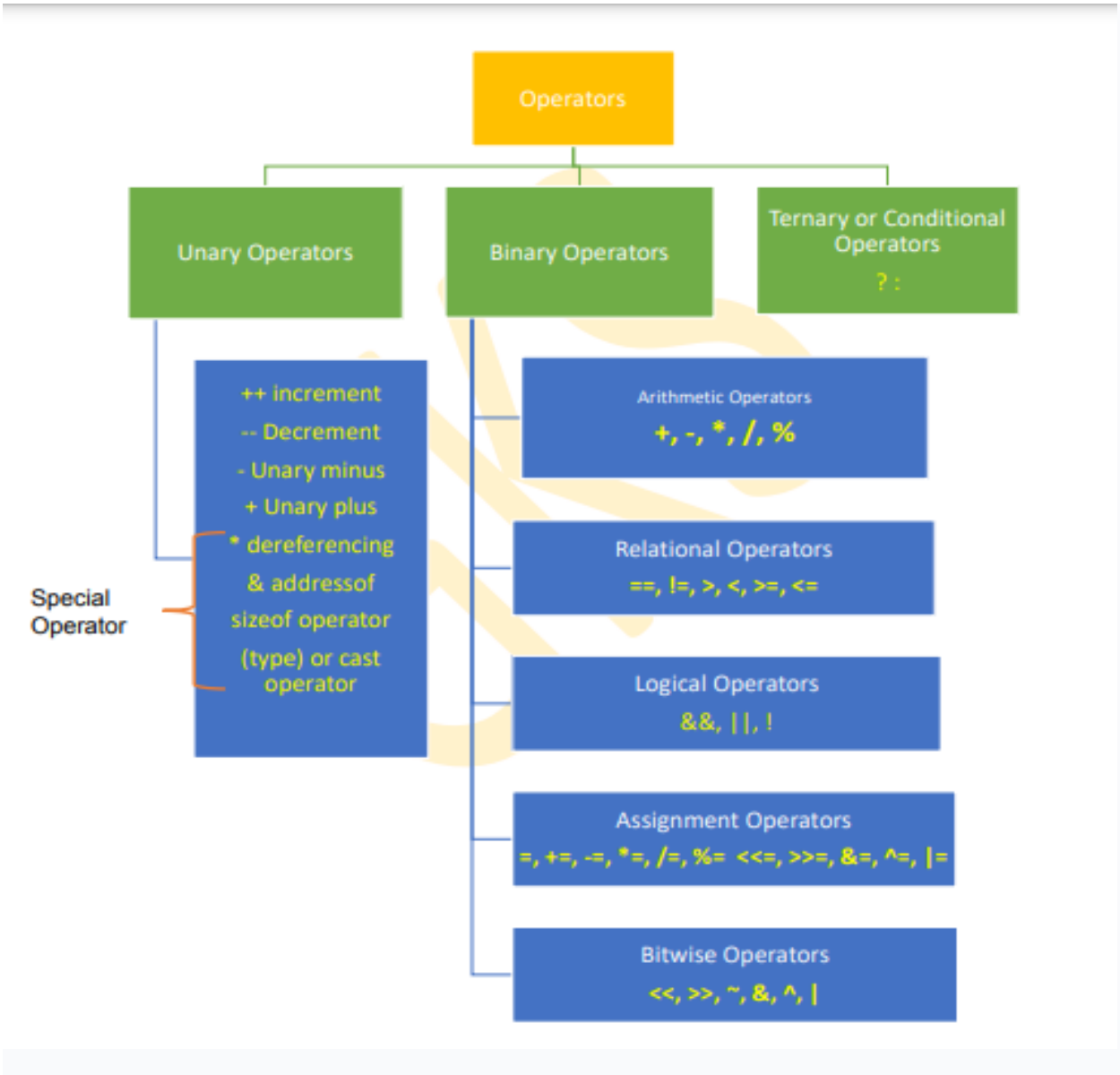
Contents:

C Operators - Arithmetic operators, relational operators, and logical operators, assignment operators, increment and decrement operators, conditional operators, special operators, arithmetic expressions, evaluation of expressions, precedence of arithmetic operators, Type conversion in expressions, operator precedence and associativity, Mathematical Functions, I/O operations - Library functions.

Operators In C

An operator is a symbol that operates on a value or a variable. For example: + is an operator to perform addition.

C has a wide range of operators to perform various operations.



C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Example 1: Arithmetic Operators

```
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, 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 = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

The operators +, - and * computes addition, subtraction, and multiplication respectively as you might have expected.

In normal calculation, 9/4 = 2.25. However, the output is 2 in the program.

It is because both the variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after the decimal point and shows answer 2 instead of 2.25.

The modulo operator % computes the remainder. When a=9 is divided by b=4, the remainder is 1. The % operator can only be used with integers.

Suppose a = 5.0, b = 2.0, c = 5 and d = 2. Then in C programming,

```
// Either one of the operands is a floating-point number
```

```
a/b = 2.5
a/d = 2.5
c/b = 2.5

// Both operands are integers
c/d = 2
```

C Increment and Decrement Operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

Output

```
++a = 11
--b = 99
++c = 11.500000
--d = 99.500000
```

Here, the operators ++ and -- are used as prefixes. These two operators can also be used as postfixes like a++ and a--. Visit this page to learn more about how increment and decrement operators work when used as postfix.

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b

/=	a /= b	a = a/b
%=	a %= b	a = a%b

Example 3: Assignment Operators

```
// Working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;      // c is 5
    printf("c = %d\n", c);
    c += a;     // c is 10
    printf("c = %d\n", c);
    c -= a;     // c is 5
    printf("c = %d\n", c);
    c *= a;     // c is 25
    printf("c = %d\n", c);
    c /= a;     // c is 5
    printf("c = %d\n", c);
    c %= a;     // c = 0
    printf("c = %d\n", c);

    return 0;
}
```

Output

```
c = 5
c = 10
c = 5
```

```
c = 25
c = 5
c = 0
```

C Relational Operators

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.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Example 4: Relational Operators

```
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Output

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
```

```
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

Example 5: Logical Operators

```
// Working of logical operators

#include <stdio.h>
```

```
void main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) is %d \n", result);

   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) is %d \n", result);

   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) is %d \n", result);

   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) is %d \n", result);

   result = !(a != b);
   printf("!(a != b) is %d \n", result);

   result = !(a == b);
   printf("!(a == b) is %d \n", result);


}
```

Output

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

Explanation of logical operator program

- (a == b) && (c > 5) evaluates to 1 because both operands (a == b) and (c > b) is 1 (true).
- (a == b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).
- (a == b) || (c < b) evaluates to 1 because (a = b) is 1 (true).
- (a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).
- !(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).
- !(a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false).

C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

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

The sizeof operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example 6: sizeof Operator

```
#include <stdio.h>
int main()
{
   int a;
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));

   return 0;
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
```

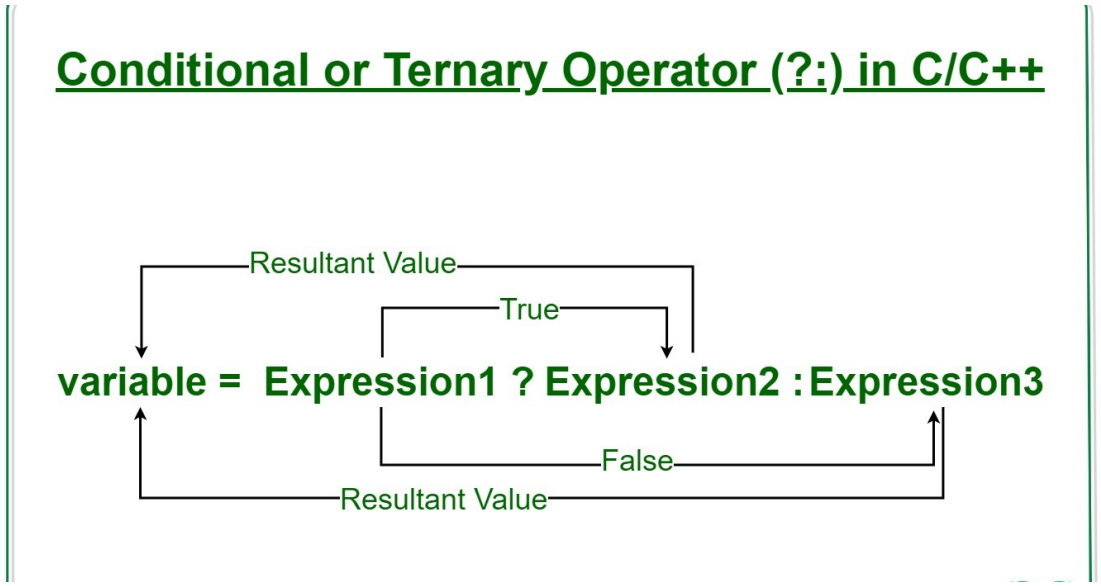
Ternary Operator in C

- If any operator is used on three operands or variable is known as **Ternary Operator**.
- It can be represented with **? :** . It is also called as **conditional operator**
- It helps to think of the ternary operator as a shorthand way or writing an if-else statement.
- Programmers use the ternary operator for decision making in place of longer if and else conditional statements.
- The ternary operator is an operator that takes three arguments.
- The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison. Ternary operator is shortened way of writing an if-else statement.

Ternary operator is **a?b:c** it say that the condition a is true b will be executed else c will be executed.

Syntax

expression-1 ? expression-2 : expression-3



Example

Find Largest Number among 3 Number in C

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

void main()
{
int a, b, c, large;
clrscr();
printf("Enter any three number: ");
scanf("%d%d%d",&a,&b,&c);
large=a>b ? (a>c?a:c) : (b>c?b:c);
printf("Largest Number is: %d",large);
getch();

}
```

Output

```
Enter any three number: 5 7 2

Largest number is 7
```

Expression Evaluation in C

- In c language expression evaluation mainly depends on priority and associativity.
- An expression is a sequence of operands and operators that reduces to a single value. For example, the expression, 10+15 reduces to the value of 25.
- An expression is a combination of variables constants and operators written according to the syntax of C language.
- every expression results in some value of a certain type that can be assigned to a variable.

Operator precedence

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Priority

This represents the evaluation of expression starts from "what" operator.

Associativity

It represents which operator should be evaluated first if an expression is containing more than one operator with same priority.

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Example Code

```c
#include <stdio.h>

main() {

   int a = 20;

   int b = 10;

   int c = 15;

   int d = 5;

   int e;

   e = (a + b) * c / d; // ( 30 * 15 ) / 5

   printf("Value of (a + b) * c / d is : %d\n", e );

   e = ((a + b) * c) / d; // (30 * 15 ) / 5

   printf("Value of ((a + b) * c) / d is : %d\n" , e );

   e = (a + b) * (c / d); // (30) * (15/5)

   printf("Value of (a + b) * (c / d) is : %d\n", e );

   e = a + (b * c) / d; // 20 + (150/5)

   printf("Value of a + (b * c) / d is : %d\n" , e );

   return 0;
```

```
}
```

Output

```
Value of (a + b) * c / d is : 90

Value of ((a + b) * c) / d is : 90

Value of (a + b) * (c / d) is : 90

Value of a + (b * c) / d is : 50
```

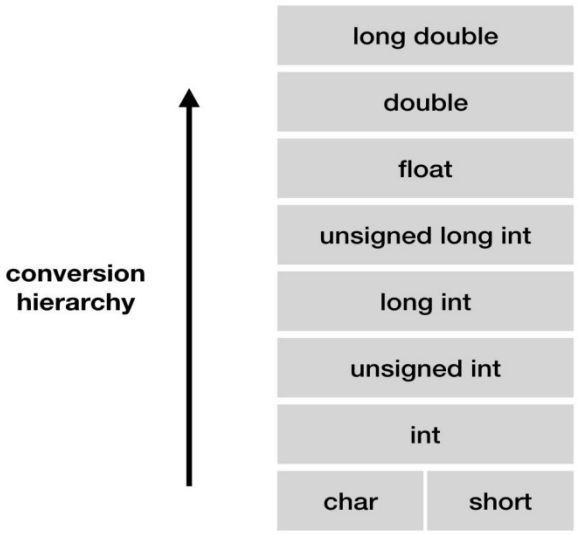
Type Conversion in C

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1.Implicit Type Conversion

2.Explicit Type Conversion

1.implicit Type conversion



- Also known as ‘automatic type conversion’.
- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid loss of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.

bool -> char -> short int -> int -> unsigned int -> long -> unsigned ->

long long -> float -> double -> long double

- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Example of Type Implicit Conversion:

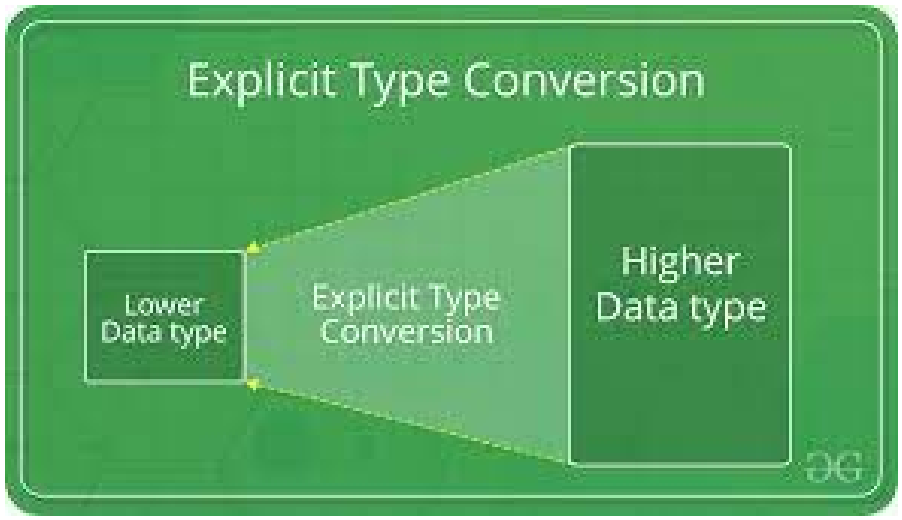
```
// An example of implicit conversion

#include<stdio.h>

int main()

{

   int x = 10;    // integer x

   char y = 'a';  // character c



   // y implicitly converted to int. ASCII

   // value of 'a' is 97

   x = x + y;



   // x is implicitly converted to float

   float z = x + 1.0;
```

```
    printf("x = %d, z = %f", x, z);

    return 0;

}
```

Output:

x = 107, z = 108.000000

2. **Explicit Type Conversion**



This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

```
(type) expression
```

Type indicated the data type to which the final result is converted.

Exapmle:

// C program to demonstrate explicit type casting

```
#include<stdio.h>

int main()
{
        double x = 1.2;

        // Explicit conversion from double to int
        int sum = (int)x + 1;

        printf("sum = %d", sum);

        return 0;
}
```

Output:

sum = 2

Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.

Mathematical Functions in C

- C Programming allows us to perform mathematical operations through the functions defined in **<math.h>** header file.
- The <math.h> header file contains various methods for performing mathematical operations such as sqrt(), pow(), ceil(), floor() etc.

C Math Functions

There are various methods in math.h header file. The commonly used functions of math.h header file are given below.

No.	Function	Description
1)	ceil(number)	rounds up the given number. It returns the integer value which is greater than or equal to given number.
2)	floor(number)	rounds down the given number. It returns the integer value which is less than or equal to given number.
3)	sqrt(number)	returns the square root of given number.
4)	pow(base, exponent)	returns the power of given number.
5)	abs(number)	returns the absolute value of given number.

C Math Example

Let's see a simple example of math functions found in math.h header file.

```c
#include<stdio.h>
#include <math.h>
int main(){
printf("\n%f",ceil(3.6));
printf("\n%f",ceil(3.3));
printf("\n%f",floor(3.6));
printf("\n%f",floor(3.2));
printf("\n%f",sqrt(16));
printf("\n%f",sqrt(7));
printf("\n%f",pow(2,4));
printf("\n%f",pow(3,3));
printf("\n%d",abs(-12));
 return 0;
}
```

Output:

```
4.000000
4.000000
3.000000
3.000000
4.000000
2.645751
16.000000
27.000000
12
```

Managing Input and Output Operations

- Reading, processing, and writing of data are the three essential functional functions of a computer program input and output operations.
- Unlike other high-level languages, C does not have any built-in input/output statements as part of its syntax. All input/output operations are carried out through function calls such as **printf** and **scanf**.
- There exist several functions that have more or less become standard for input and output operations in C. These functions are collectively known as the standard input and output library.

Reading a Character

The simplest of all managing input and output operations in c is reading a character from the 'standard input' unit (usually the keyboard) and writing it to the 'standard output' unit (usually the screen). Reading a single character can be done by using the function getchar.

Syntax

```
variable_name=getchar();
```

The program displays a question of YES/NO type to the user and reads the user's response in a single character (Y or N). If the response is Y or Y, it outputs the message **My name is BUSY BEE**

```
Program
  #include <stdio.h>
  main()
  {
     char answer;
     printf("Would you like to know my name?\n");
     printf("Type Y for YES and N for NO: ");
     answer = getchar(); /* .... Reading a character...*/
     if(answer == 'Y' || answer == 'y')
        printf("\n\nMy name is BUSY BEE\N");
     else
     printf("\n\nYou are good for nothing\n");
  }
```

Output

```
     Would you like to know my name?
     Type Y for YES and N for NO: Y
     Would you like to know my name?
```

```
    Type Y for YES and N for NO: n

    You are good for nothing
```

The **getchar** function may be called successively to read the characters contained in a line of text. For example, the following program segment reads characters from keyboard one after another untile the 'Return' key is pressed.

The **getchar()** function accepts any character keyed in. This includes RETURN and TAB . This means when we enter single character input, the newline character is waiting in the input queue after **getchar()** returns.

Writing a Character – input and output

Like getchar, there is an analogous function putchar for writing characters one at a time to the terminal. It takes the form as shown below:

Syntax:

putchar (variable_name);

where variable_name is a type char variable containing a character. This statement display the character contained in the variable_name at the terminal.

Eg:

```
answer = 'Y' ;
```

```
putchar (answer);
```

Formatted Input

The control string specifies the field format in which the data is to be entered and the argument arg1, arg2,... argn specify the address of locations where the data is stored, Control string and arguments are separted by commas.

- Field (or format) specifications, consisting of the conversion character % a data type character (or type specifier), and an optional number, specifying the field width.
- Blanks, tabs, or newlines.
- Blanks, tabs and newlines are ignoerd. The data type character indicates the type of data is to be assigned to the variable associated with the corresponding argument. The field width specifier is optional. The discussions that follo will clarify these concepts.

Inputting Integer Numbers – input and output

The field specification for reading an integer number is : ***% w sd***

The percentage sign (%) indicates that a conversion specification follows. w is an integer number that specified the field width of the number to be read and d, know as data type character, indicates that the number to be read is in interger mode. Consider the following example: **scanf ("%d %5d", &num1, &num2);**

The variable num1 will be assigned 31 (because of %2d) and num2 will be assigned 426 (unread part of 31426). The value 50 that is unread will be

assigned to the first variable in the next scanf call. This kind of error may be eliminated if we use the field specifications without the without the field width specifications. That is the statement **scanf("%d %d", &num1, &num2);**

Points to Remember While Using scanf – input and output

If we do not plan carefully, some 'crazy' things can heppen with scanf. Since the Input and output are not part of C language, they are made available either as a separate module of the C library or as a part of the operating system (like UNIX). New feature are added to these routines from time to time as never versions of systems are released. We should consult the system referance manual before using these routines.

1. All function arguments, except the control string must be pointers to variables.
2. Format specifications contained in the control string should match the arguments in order.
3. Input data items must be separated by spaces and must match the variables reciving the input in the same order.
4. The reading will be terminated, when **scanf** encounters a 'mismatch' of data or a character that is not valid for value being read.
5. When searching for a value, **scanf** ignores line boundaries and simply looks for the next appropriate character.
6. Any under data items in a line be considered as part of the data input to the next scanf call.
7. When the field width specifier w is used, it should be large enough to contain the input data size.

Rules for scanf

- Each variable to be read must have a filed have a specification.
- For each field specification, there must be a variable address of proper type.
- Any non-whitespace character used in the formated string must have a matching character in the user input.
- Never end the format string with whitespace. It is a fatal error!

Simple C programs

Program to Display "Hello, World!"

```
#include <stdio.h>
Void main() {
   // printf() displays the string inside quotation
   printf("Hello, World!");

}
```

Output

```
Hello, World!
```

Program to Print an Integer

```
#include <stdio.h>
int main() {
    int number;

    printf("Enter an integer: ");

    // reads and stores input
    scanf("%d", &number);

    // displays output
```

```
    printf("You entered: %d", number);

    return 0;
}
```

Output

```
Enter an integer: 25
You entered: 25
```

Program to Add Two Integers

```
#include <stdio.h>
int main() {

    int number1, number2, sum;

    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    // calculating sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    return 0;
}
```

Output

```
Enter two integers: 12
11
12 + 11 = 23
```

Program to Multiply Two Numbers

```
#include <stdio.h>
int main() {
    double a, b, product;
```

```
    printf("Enter two numbers: ");
    scanf("%lf %lf", &a, &b);

    // Calculating product
    product = a * b;

    // %.2lf displays number up to 2 decimal point
    printf("Product = %.2lf", product);

    return 0;
}
```

Output

```
Enter two numbers: 2.4
1.12
Product = 2.69
```