C PROGRAMMING

FYBSC SEM 2

INTRODUCTION

- A general purpose language, closely associated with UNIX that developed in BELL laboratories.
- Most of the programs of UNIX is written with the help of C.
- C was originally developed at Bell Labs by Dennis Ritchie between 1972 and 1973 to construct utilities running on Unix.
- From beginning C was useful for busy programmers to get things done easily because C is powerful, dominant and flexible language.

About C

- It is a structured and disciplined approach to program design.
- It supports functions that enables easy maintainability of code by breaking large file into smaller modules.
- Comments in C provides easy readability.
- The C program is built from:

1. Variables and type declarations.
2. Functions
3. Statements.
4. Expressions.

Cont.

- C language is also known as “Compiled Language” as the source code must first be compiled in order to run.

Structure of C program

1. Documentation Section.
2. Link Section.
3. Global Declaration section.
4. Main()

{

Declaration part1

Declaration part2

}

5. Subprogram section.

Function 1

Function 2

.

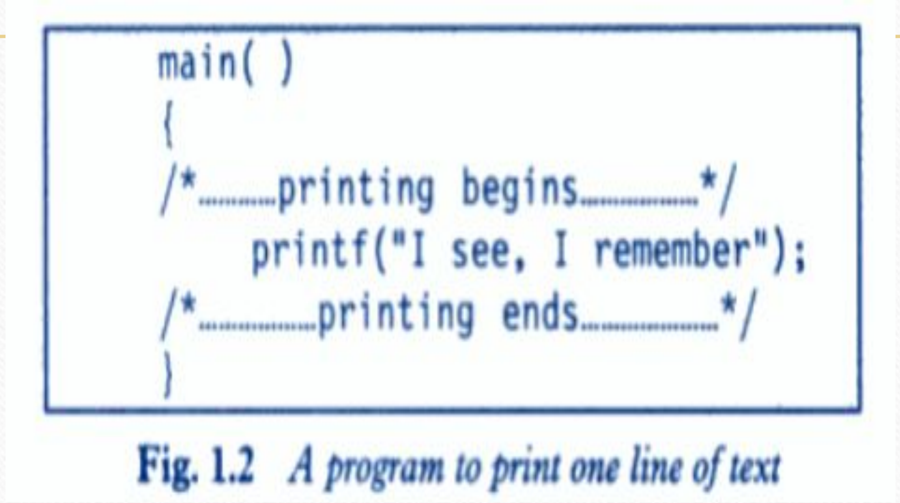
.

Function n

Header and Body.

- The **header** includes the name of the function and tells us (and the compiler) what type of data it expects to receive (the *parameters*) and the type of data it will return (*return value type*) to the calling function or program.
- Eg: #include<stdio.h>
- The body of the function contains the instructions to be executed.

Basic syntax



```
main( )
{
/*........printing begins..............*/
    printf("I see, I remember");
/*..............printing ends...............*/
}
```

Fig. 1.2 *A program to print one line of text*

Use of Comments

- The comments are basic simple lines which are just used to read the code or understand the code.
- This comment lines are not executed by the compiler as it is only for reading purpose only.
- These are used to enhance its readability and understanding.
- There are two types of comment line:

1. Single line comment.
2. Multi line comment.

Cont.

1. A single line comment is used to comment a single line.

Eg: //Single line comment.

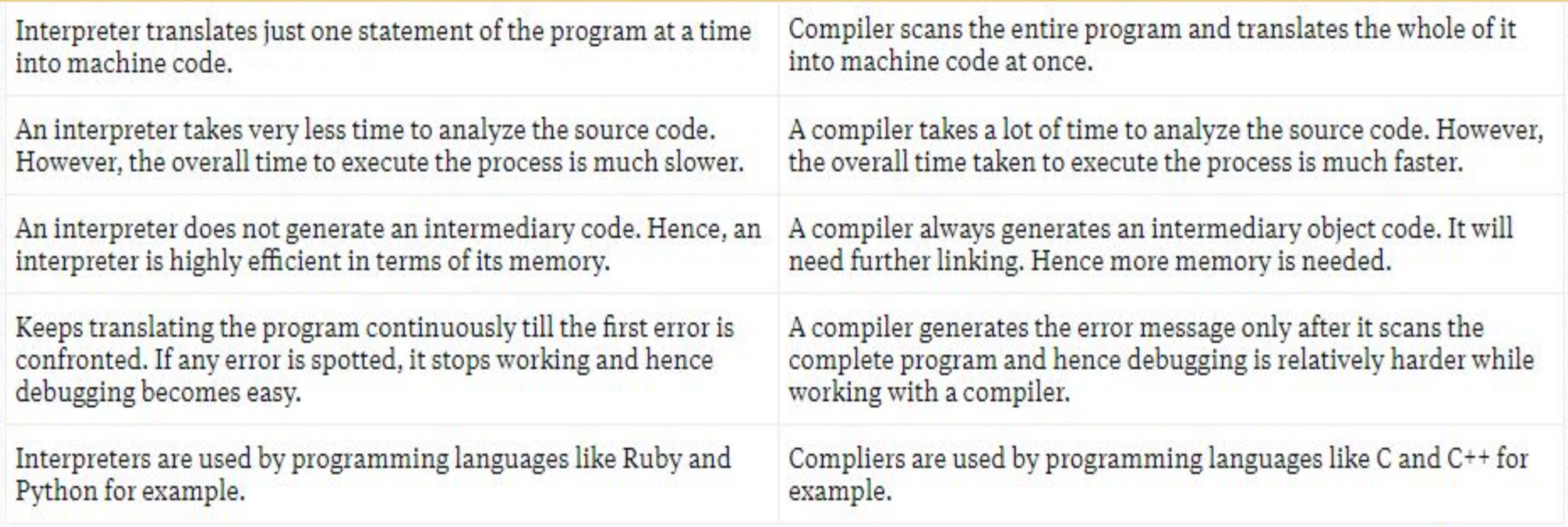
2. A multi-line comment is used to comment multiple lines in code.

```
Eg: /* this is multi line
          .
          .
    comment ends */
```

Interpreters Vs Compiler

- Compliers and interpreters are programs that help convert the high level language (Source Code) into machine codes to be understood by the computers.
- A high level language is one that can be understood by humans.
- However, computers cannot understand high level languages as we humans do. So, the source codes must be converted into machine language and here comes the role of compilers and interpreters.

Cont.

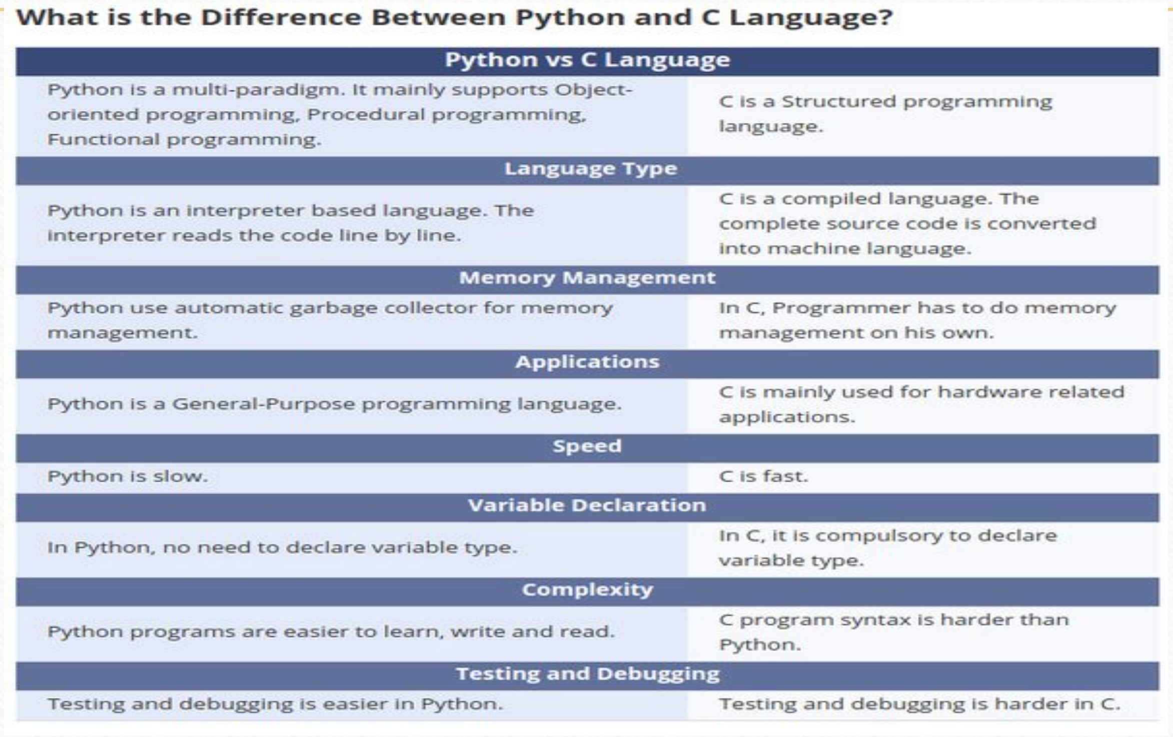


Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.
An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower.	A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster.
An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.
Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy.	A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.
Interpreters are used by programming languages like Ruby and Python for example.	Compliers are used by programming languages like C and C++ for example.

Python Vs C

- Python is a general-purpose, high-level programming language that was developed by Guido Rossum in 1989. What makes Python amazing is its simple syntax that is almost similar to the English language and dynamic typing capability. The straightforward syntax allows for easy code readability.
- Being an interpreted language, Python is an ideal language for scripting and rapid application development on most platforms and is so popular with the developers. Scripting languages incorporate both interactive and dynamic functionalities via web-based applications.

Cont.



What is the Difference Between Python and C Language?

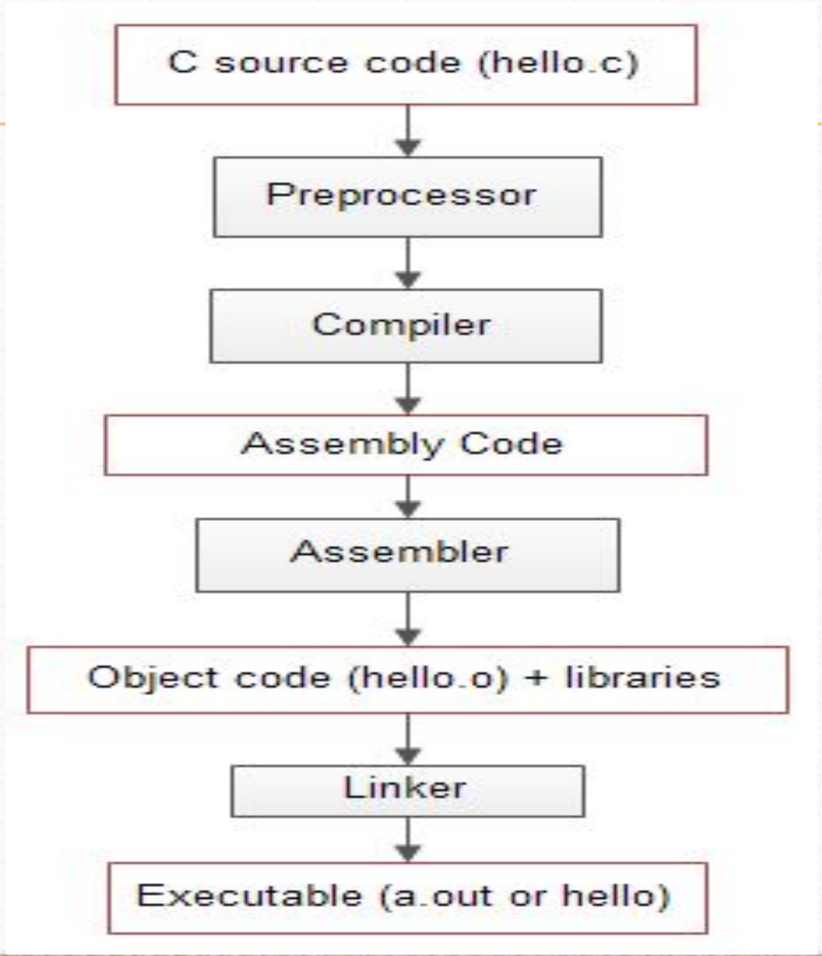
Python vs C Language	
Python is a multi-paradigm. It mainly supports Object-oriented programming, Procedural programming, Functional programming.	C is a Structured programming language.
Language Type	
Python is an interpreter based language. The interpreter reads the code line by line.	C is a compiled language. The complete source code is converted into machine language.
Memory Management	
Python use automatic garbage collector for memory management.	In C, Programmer has to do memory management on his own.
Applications	
Python is a General-Purpose programming language.	C is mainly used for hardware related applications.
Speed	
Python is slow.	C is fast.
Variable Declaration	
In Python, no need to declare variable type.	In C, it is compulsory to declare variable type.
Complexity	
Python programs are easier to learn, write and read.	C program syntax is harder than Python.
Testing and Debugging	
Testing and debugging is easier in Python.	Testing and debugging is harder in C.

Compilation

- The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.
- The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.
- In the pre-processor, The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

- In the compiler, The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.
- Then the assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. If the name of the source file is **'hello.c',** then the name of the object file would be 'hello.obj'.
- The linking is the process of putting together other program files and functions that are required by the program. For eg: if the program is using exp() function then the object code of this function should be brought from the math library.

Diagram.


C source code (hello.c)
Preprocessor
Compiler
Assembly Code
Assembler
Object code (hello.o) + libraries
Linker
Executable (a.out or hello)

Formatted I/O

- C provides standard functions scanf() and printf(), for performing formatted input and output .These functions accept, as parameters, a format specification string and a list of variables.
- The format specification string is a character string that specifies the data type of each variable to be input or output and the size or width of the input and output.

For Input

- Scanf(): To read data in from standard input (keyboard), we call the **scanf** function. The basic form of a call to scanf is: scanf(*format_string*, *list_of_variable_addresses*);
- If **x** is a variable, then the expression **&x** means "address of x"

For Output

- Printf(): The function printf() is used for formatted output to standard output based on a format specification. The format specification string, along with the data to be output, are the parameters to the printf() function.
- Basic syntax: printf(format, data1, data2,……..);
- The character specified after % is called a conversion character because it allows one data type to be converted to another type and printed.
- The conversion character used in C are:

1. %d : the data is converted into Decimal(integer).
2. %s : The data is a string and character from the string , are printed until a NULL, character is reached.
3. %c : The data is taken as a character.
4. %f : The data is output as float or double with a default Precision 6.

Data

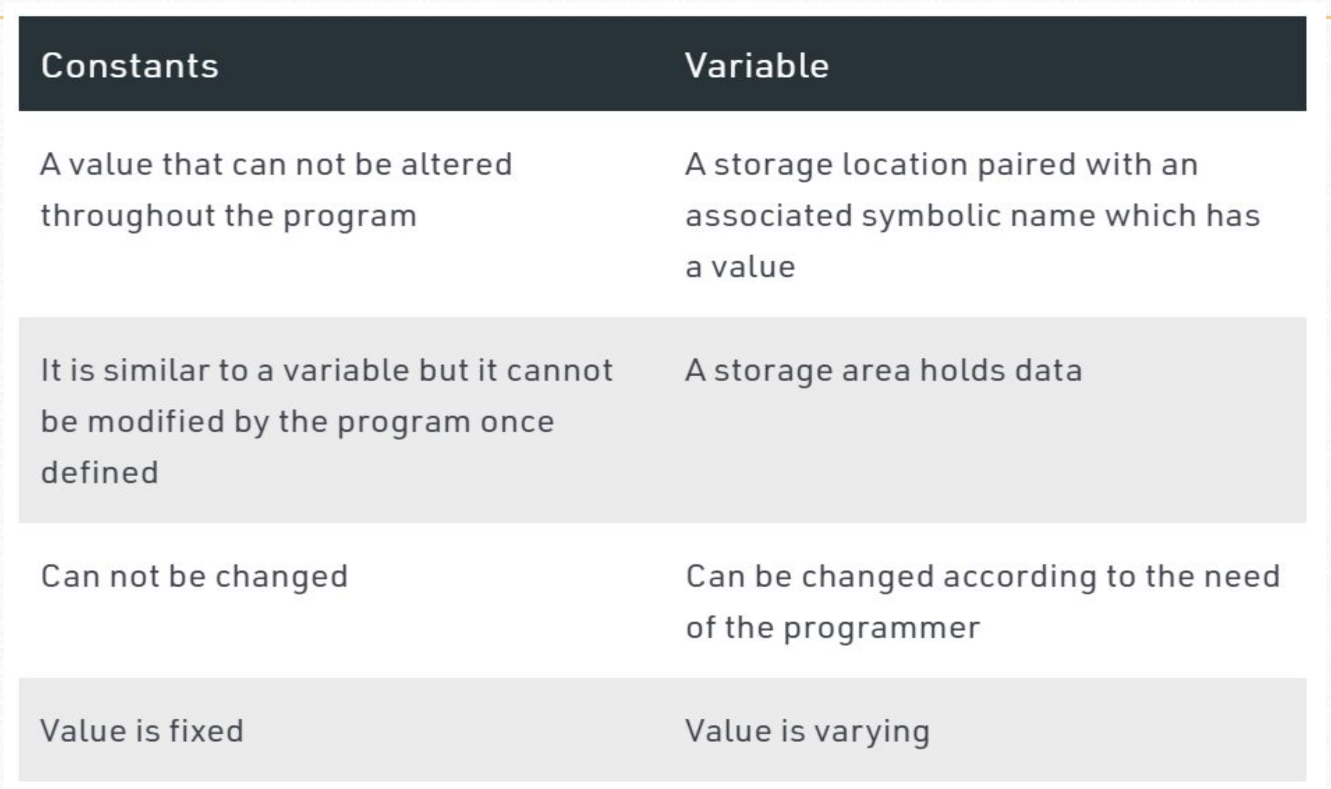
- 1. **Variables**: A variable are names used to refer to some memory locations of computer.
- The piece of information stored at this location is referred as value of variable.
- The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive.
- For eg: int a=10; (Here "int" is datatype and "a" is the variable name)

Constant

2. Constants: It is a value that cannot be changed during the execution of the program.

- Constants are treated just like regular variables except that their values cannot be modified after their definition.
- For Eg: #define PI=3.14

Difference.



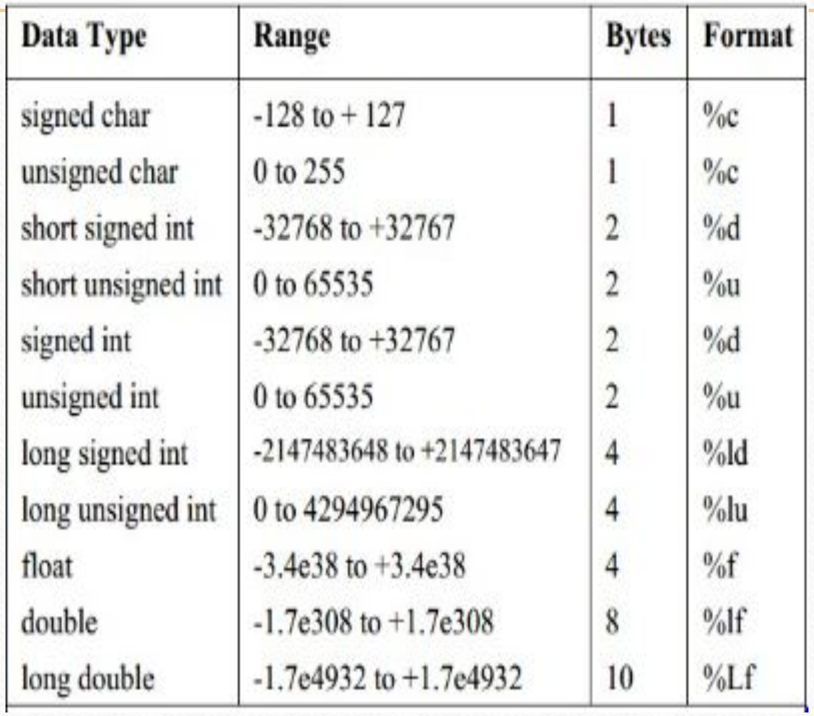
Constants	Variable
A value that can not be altered throughout the program	A storage location paired with an associated symbolic name which has a value
It is similar to a variable but it cannot be modified by the program once defined	A storage area holds data
Can not be changed	Can be changed according to the need of the programmer
Value is fixed	Value is varying

Data Types

- Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we can use in our program. These datatypes have different storage capacities.
- The data types includes are:

1. Int:
2. Float:
3. Char:
4. Void:
5. Double
6. Short
7. Long
8. Signed and unsigned

Example.



Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

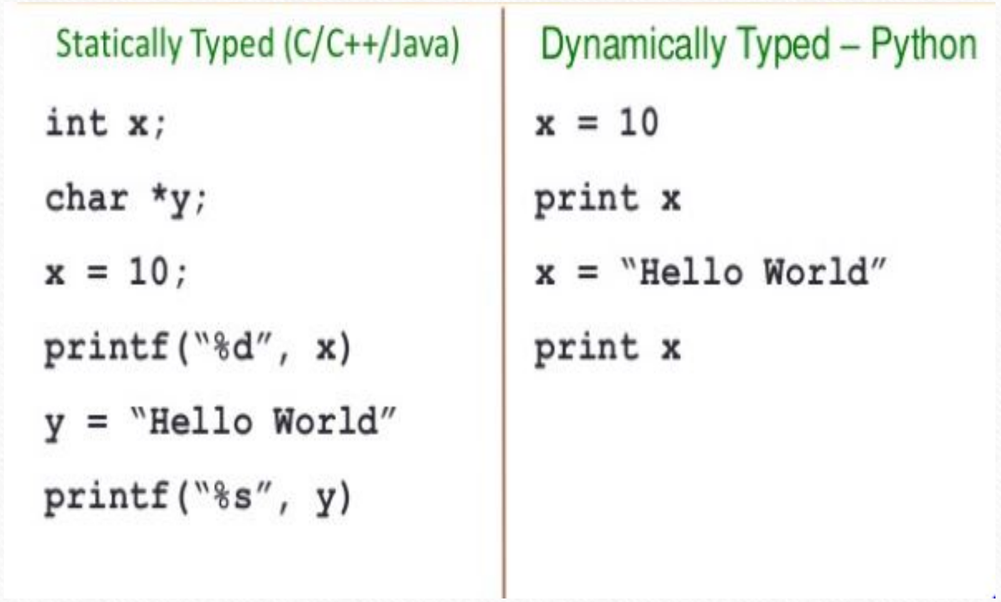
Static Typing Vs Dynamic Typing

- A language is statically-typed if the type of a variable is known at compile-time instead of at run-time. Common examples of statically-typed languages include Java, C, C++, FORTRAN, Pascal and Scala.
- In Statically typed languages, once a variable has been declared with a type, it cannot ever be assigned to some other variable of different type and doing so will raise a type error at compile-time.

Cont.

- A language is dynamically-typed if the type of a variable is checked during run-time. Common examples of dynamically-typed languages includes JavaScript, Objective-C, PHP, Python, Ruby, Lisp, and Tcl.
- In Dynamically typed languages, variables are bound to objects at run-time by means of assignment statements, and it is possible to bind the same variables to objects of different types during the execution of the program.

Difference

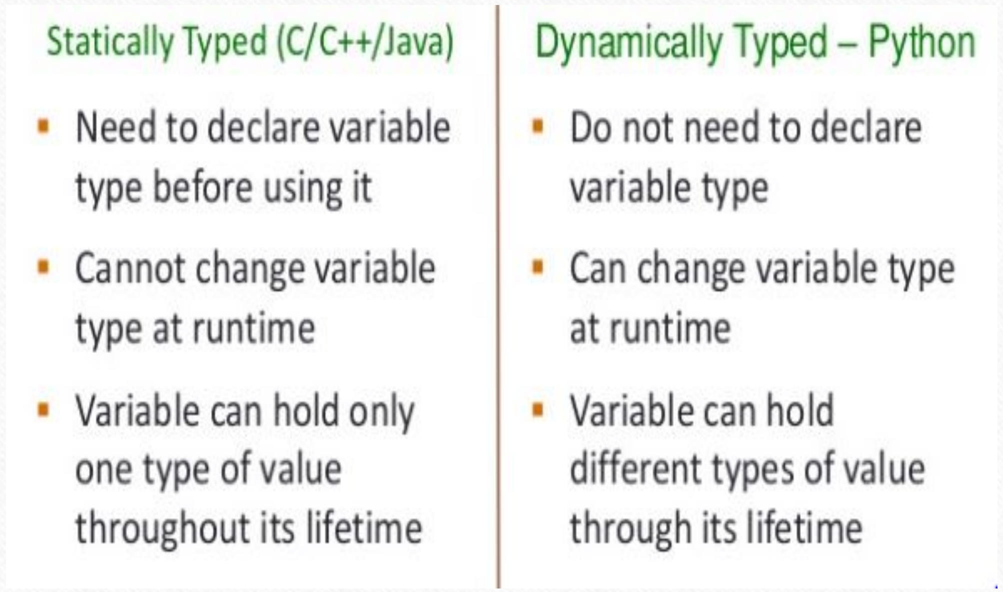


Statically Typed (C/C++/Java)

```
int x;
char *y;
x = 10;
printf("%d", x)
y = "Hello World"
printf("%s", y)
```

Dynamically Typed – Python

```
x = 10
print x
x = "Hello World"
print x
```



Statically Typed (C/C++/Java)	Dynamically Typed – Python
▪ Need to declare variable type before using it	▪ Do not need to declare variable type
▪ Cannot change variable type at runtime	▪ Can change variable type at runtime
▪ Variable can hold only one type of value throughout its lifetime	▪ Variable can hold different types of value through its lifetime

Scope of Variables

- There are Two types of Variables in C programming where you can use in the program i.e: 1. Local variable & 2. Global Variable
- **Local variable:**

1. Variables that are declare inside a function or block.
2. They can be used only by the statement that are inside that function or block code.
3. These variables are not known to functions outside.

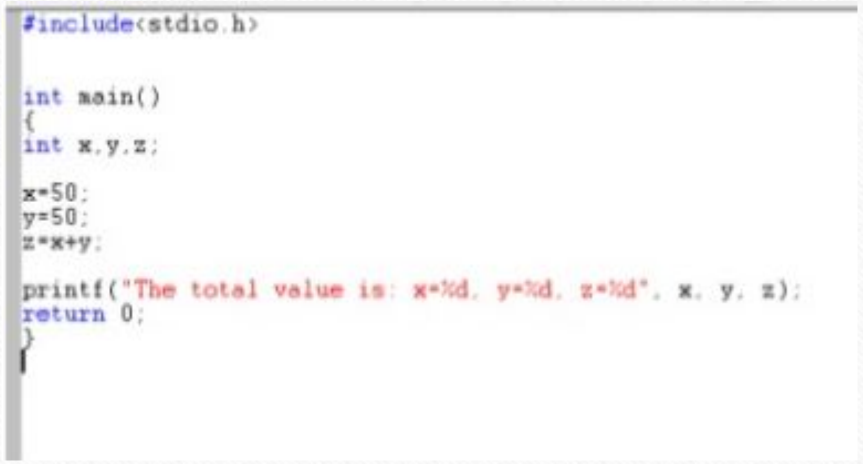
Global variable

- Global Variable:

1. Global variable are defined outside of a program, usually before the main function.
2. The global variable will hold its values throughout the program and can be accessed anywhere in the whole program.
3. It is available for the use throughout your entire program after its declaration.

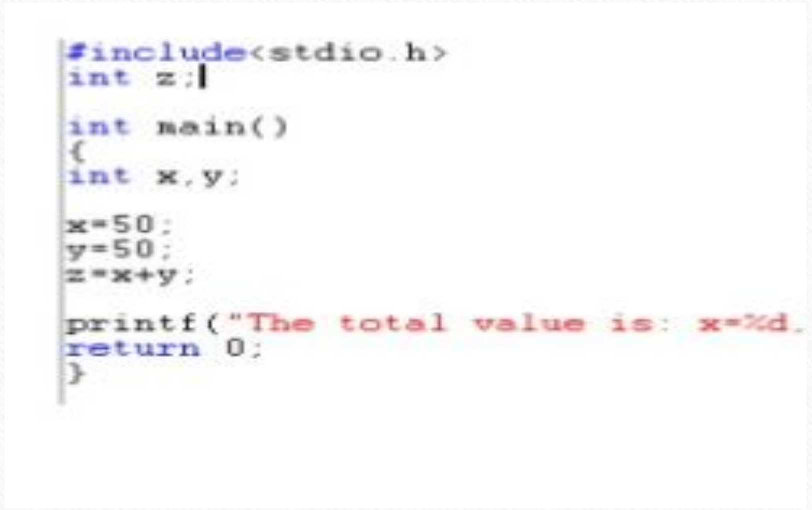
Difference

Local variable



```
#include<stdio.h>

int main()
{
int x,y,z;

x=50;
y=50;
z=x+y;

printf("The total value is: x=%d, y=%d, z=%d", x, y, z);
return 0;
}
```

Global variable

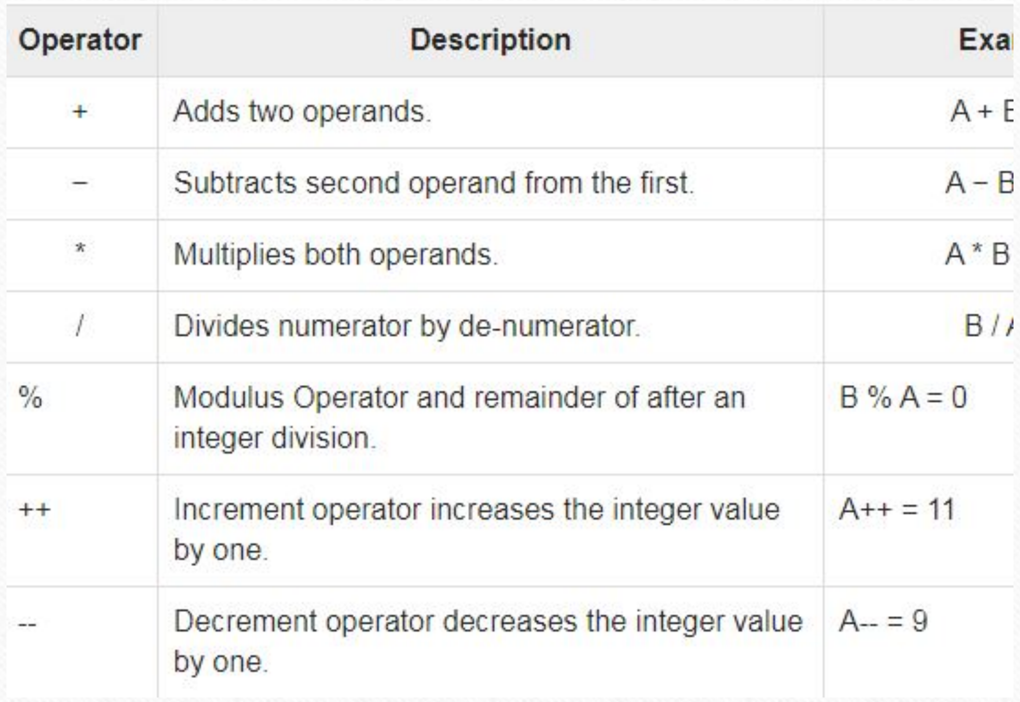


```
#include<stdio.h>
int z;

int main()
{
int x,y;

x=50;
y=50;
z=x+y;

printf("The total value is: x=%d,
return 0;
}
```

Operators

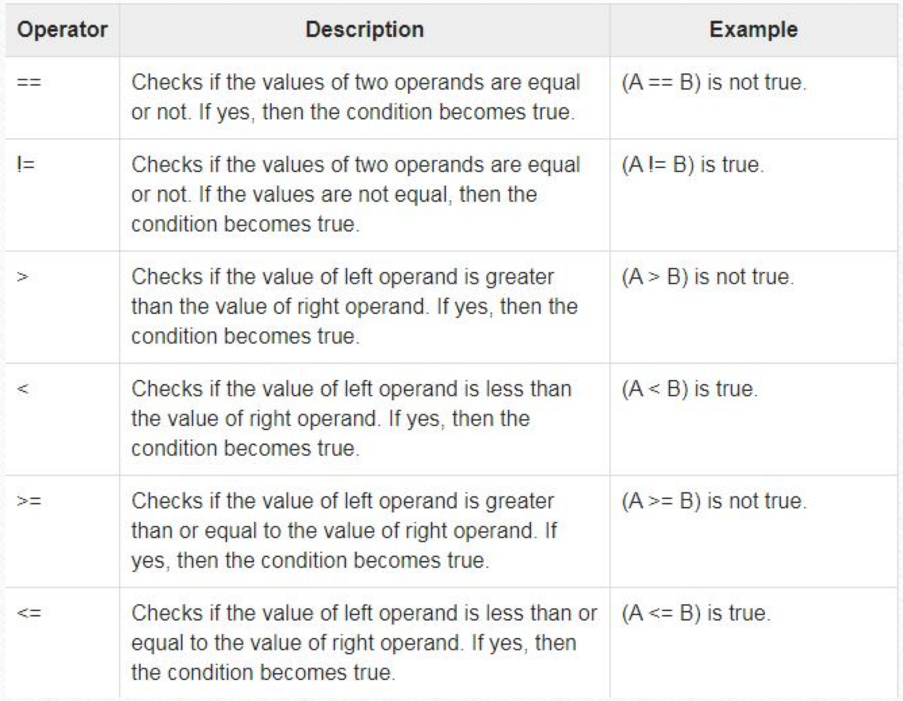
- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −
- Eg: Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators



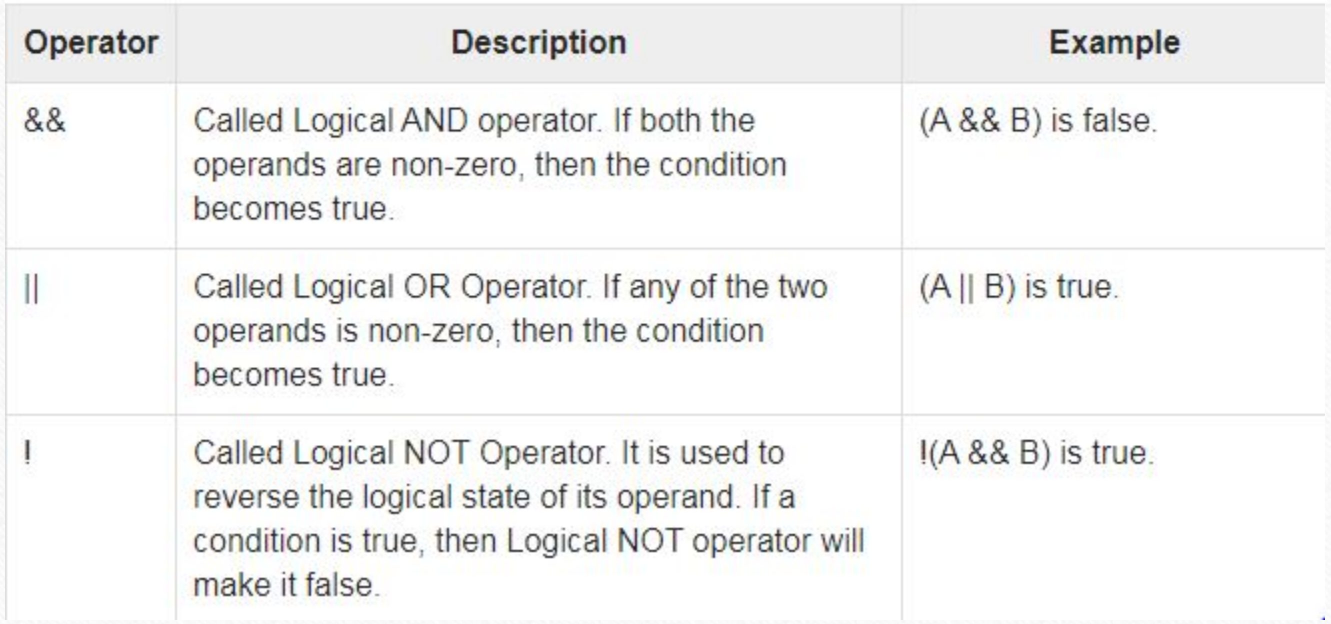
Operator	Description	Exa
+	Adds two operands.	A + B
−	Subtracts second operand from the first.	A − B
*	Multiplies both operands.	A * B
/	Divides numerator by de-numerator.	B /
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators



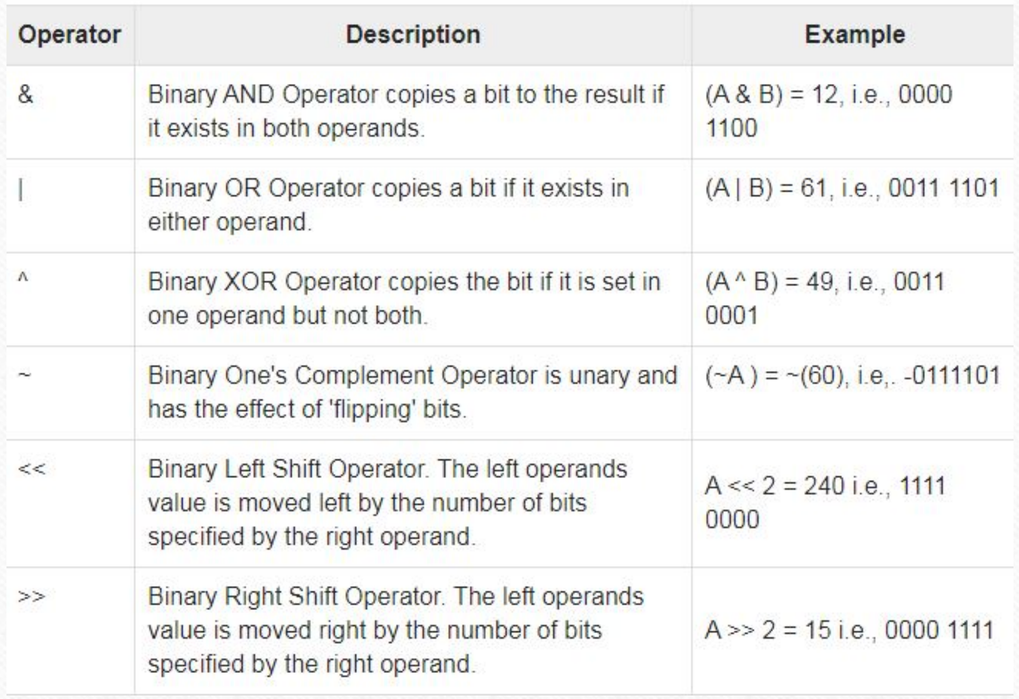
Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the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

Logical Operators



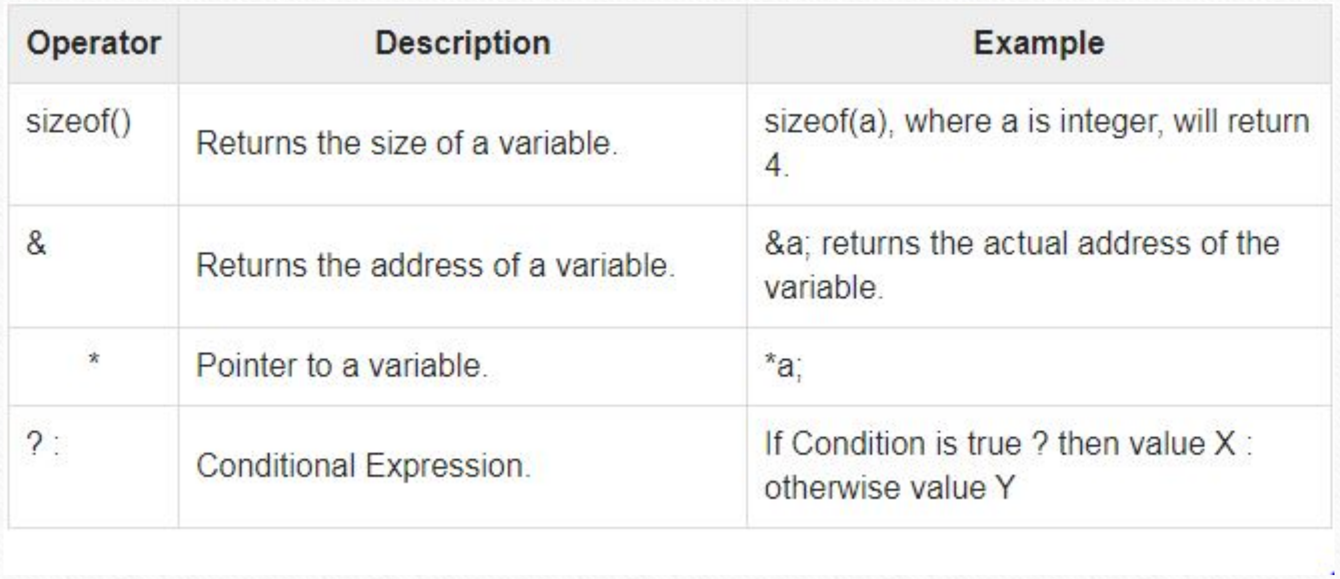
Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operator



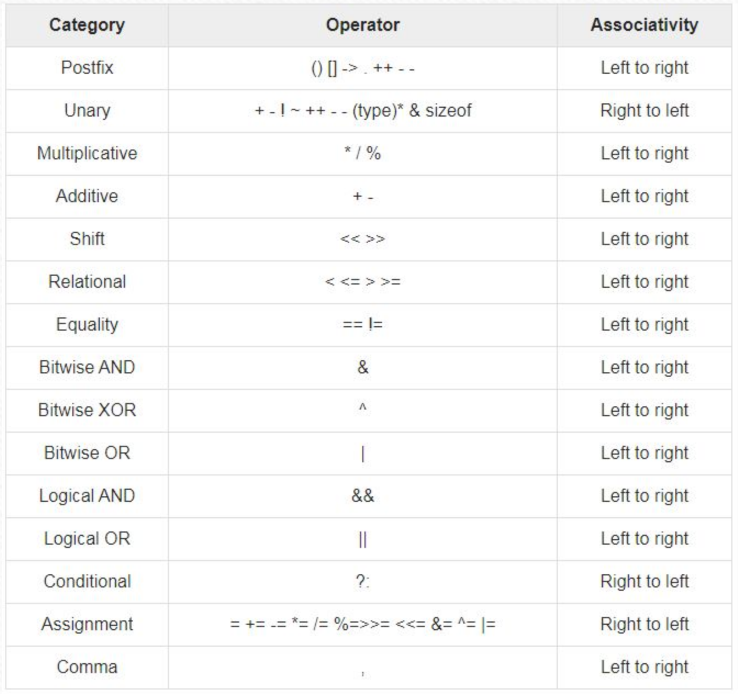
Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Misc Operators



Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Precedence in C



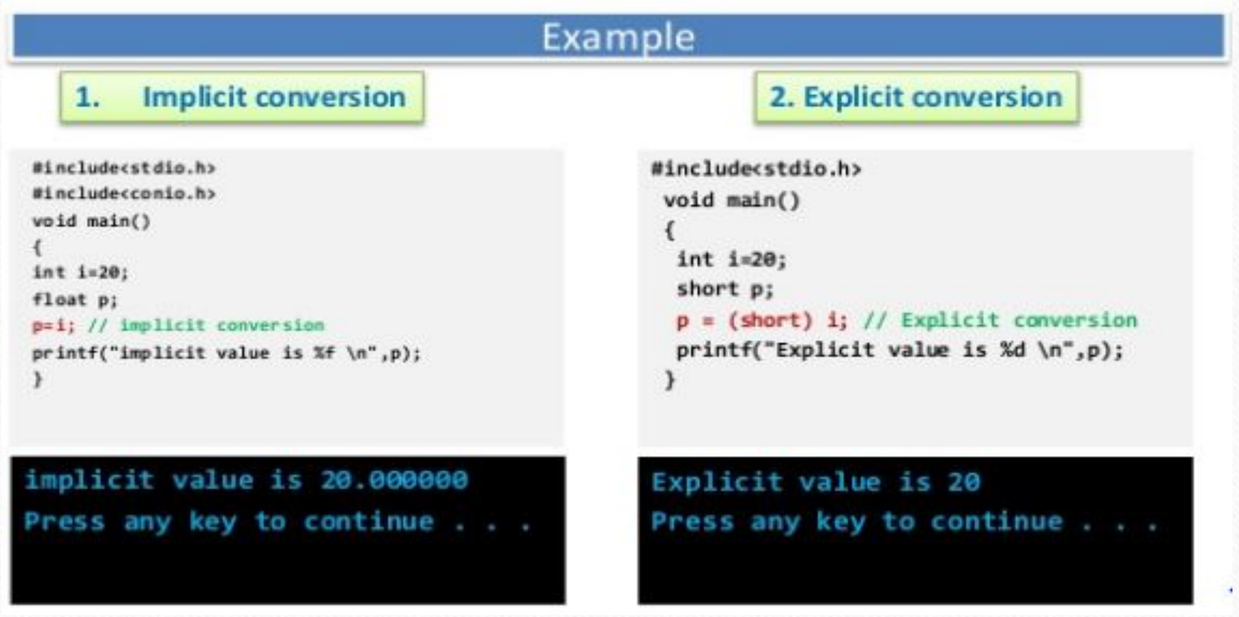
Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
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

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: Also known as ‘automatic type conversion’. Eg: int to float
2. Explicit: This process is also called type casting and it is user defined.

Example



Example

1. Implicit conversion

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i=20;
float p;
p=i; // implicit conversion
printf("implicit value is %f \n",p);
}
```

```
implicit value is 20.000000
Press any key to continue . . .
```

2. Explicit conversion

```
#include<stdio.h>
 void main()
 {
  int i=20;
  short p;
  p = (short) i; // Explicit conversion
  printf("Explicit value is %d \n",p);
 }
```

```
Explicit value is 20
Press any key to continue . . .
```

ITERATIONS

1. BRANCHING

a) if Statement

- The syntax of the if statement in C programming is:

  if (test expression) { // statements to be executed if the test expression is true }

- **How if statement works?:** The if statement evaluates the test expression inside the parenthesis ().
- If the test expression is evaluated to true, statements inside the body of if are executed.
- If the test expression is evaluated to false, statements inside the body of if are not executed.

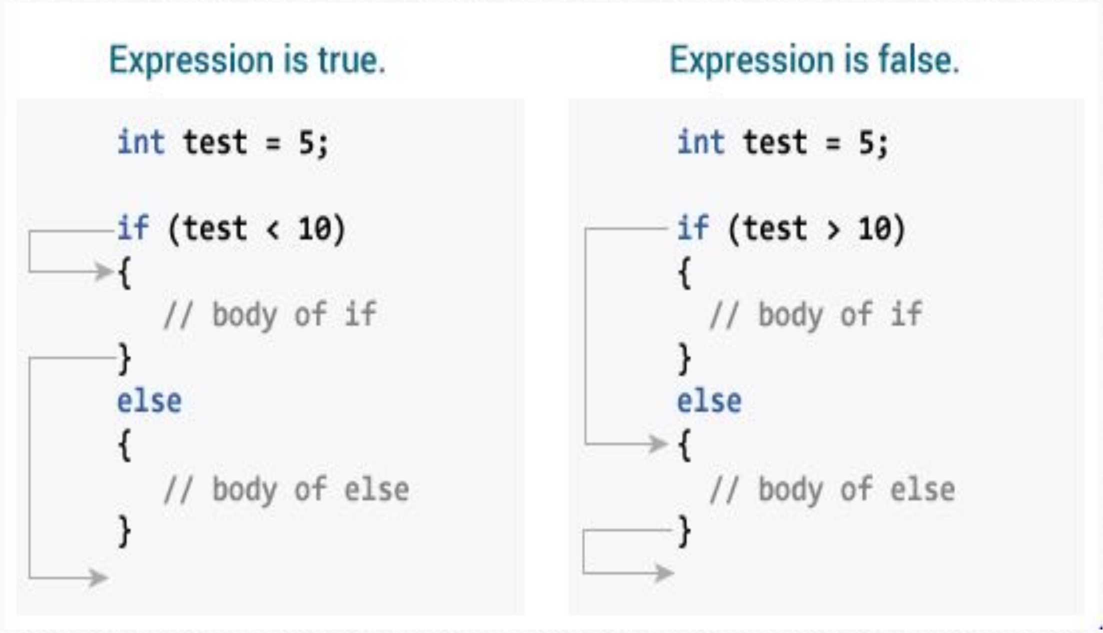
if…else statement

b) if...else Statement

- The if statement may have an optional else block. The syntax of the if..else statement is:

```
if (test expression)
{ // statements to be executed if the test expression is true }
else {
// statements to be executed if the test expression is false }
```

Example.


Expression is true.

```
int test = 5;

if (test < 10)
{
   // body of if
}
else
{
   // body of else
}
```

Expression is false.

```
int test = 5;

if (test > 10)
{
   // body of if
}
else
{
   // body of else
}
```

Switch statement

- Switch statement is a multi-way decision making statement which selects one of the several alternative based on the value of integer variable or expression.
- **Syntax:**

Switch(expression)

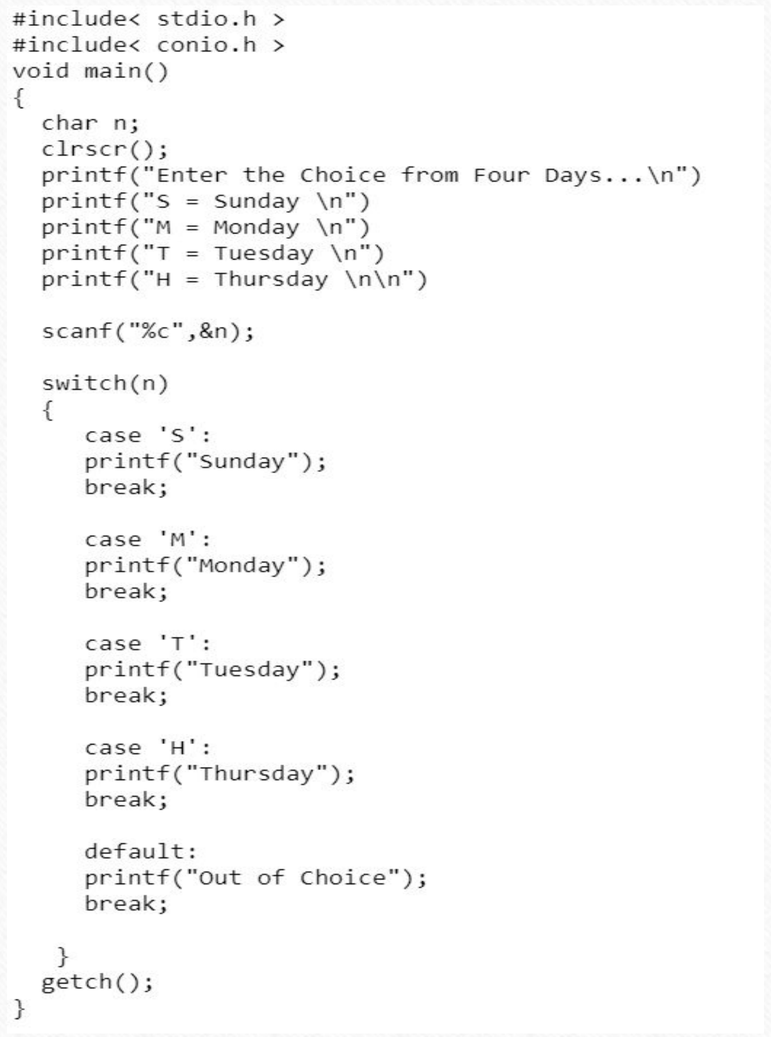
{

case constant: statement;

break;

default: statement;

}



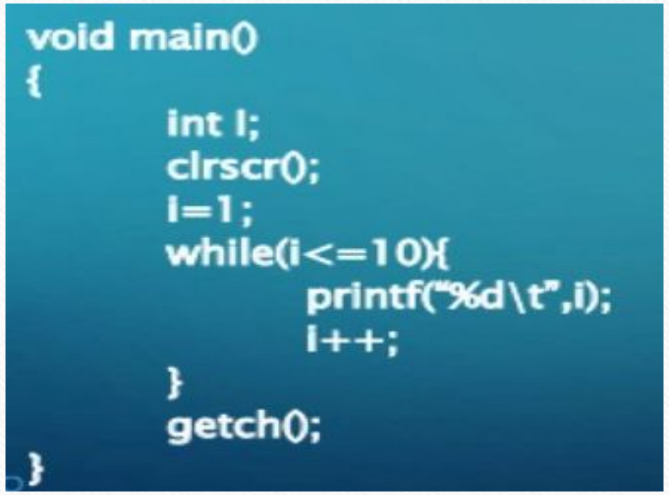
```
#include< stdio.h >
#include< conio.h >
void main()
{
  char n;
  clrscr();
  printf("Enter the Choice from Four Days...\n")
  printf("S = Sunday \n")
  printf("M = Monday \n")
  printf("T = Tuesday \n")
  printf("H = Thursday \n\n")

  scanf("%c",&n);

  switch(n)
  {
     case 'S':
     printf("Sunday");
     break;

     case 'M':
     printf("Monday");
     break;

     case 'T':
     printf("Tuesday");
     break;

     case 'H':
     printf("Thursday");
     break;

     default:
     printf("Out of Choice");
     break;

   }
  getch();
}
```

2. Looping.

a) **While loop:** this is an entry controlled looping statement. It is used to repeat a block of statements until condition becomes true.

- **Syntax**:

```
While(condition){
Statements;
Increment/decrement;
}
```

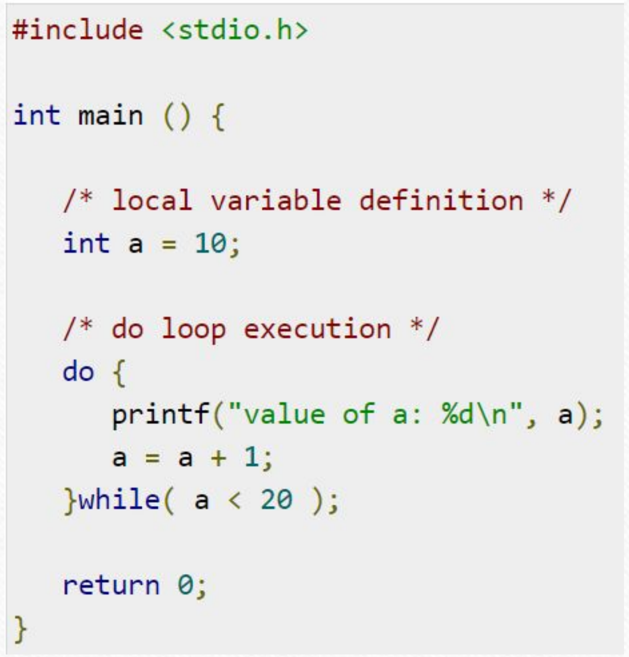


```
void main()
{
        int I;
        clrscr();
        I=1;
        while(i<=10){
                printf("%d\t",i);
                I++;
        }
        getch();
}
```

b)do…while

- This is an exit controlled looping statement. A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.
- **Syntax**

```
do {
Statement();
}while (condition);
```



```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );

   return 0;
}
```

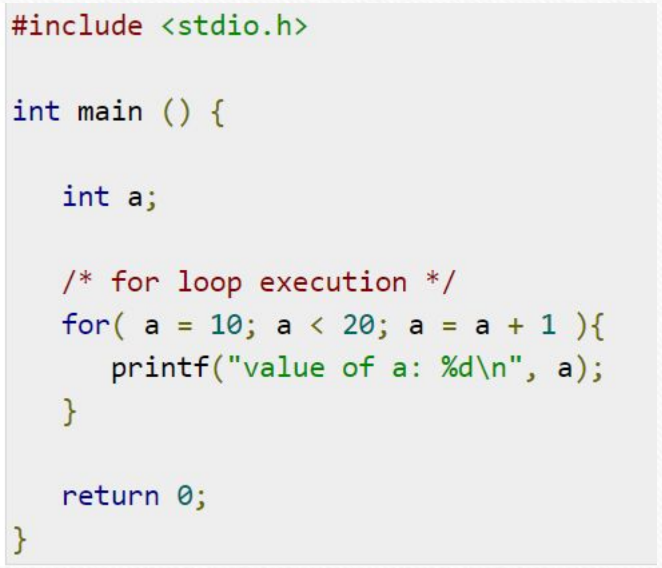
c)For loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- **Syntax**:

For (init; condition; increment){

Statement(s);

}



```c
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

3. Jump statements

A) **Break:** The **break** statement in C programming has the following two usages −

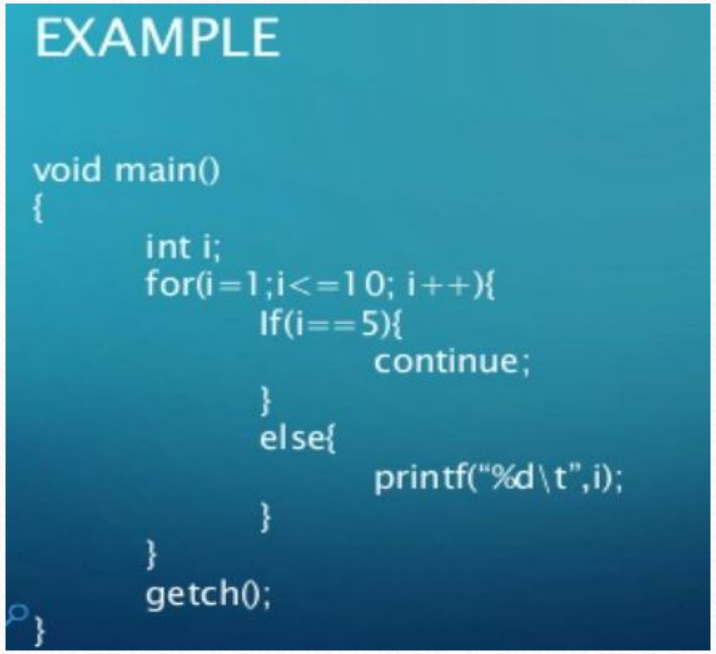
- When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement
- **Syntax:**

break;

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {

      printf("value of a: %d\n", a);
      a++;

      if( a > 15) {
         /* terminate the loop using break statement */
         break;
      }
   }

   return 0;
}
```

Continue statement

The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

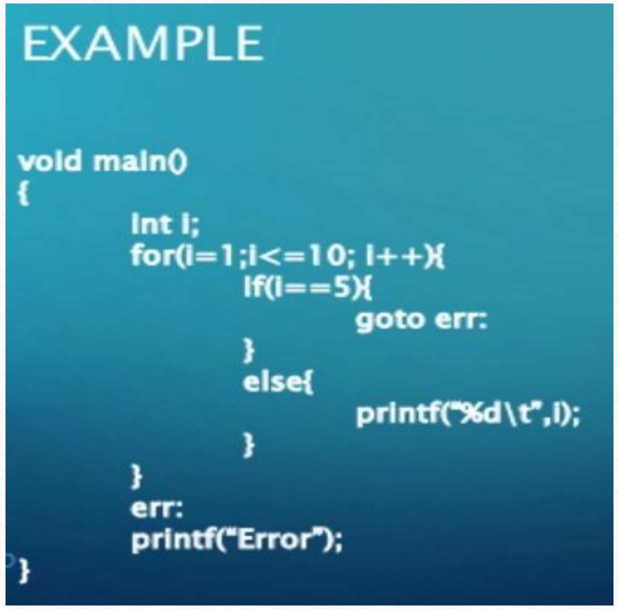


EXAMPLE

```
void main()
{
        int i;
        for(i=1;i<=10; i++){
                If(i==5){
                        continue;
                }
                else{
                        printf("%d\t",i);
                }
        }
        getch();
}
```

GOTO statement

- The goto statement is a jump statement which jump from one point to another point within a function or program.
- The goto statement is marked by label statement which can be used anywhere in the function above or below the goto statement.



EXAMPLE

```
void main()
{
        int i;
        for(i=1;i<=10; i++){
                if(i==5){
                        goto err:
                }
                else{
                        printf("%d\t",i);
                }
        }
        err:
        printf("Error");
}
```