Programming in C

Module Number: 02

Prepared By: Er. Harun

Module Name: Fundamentals of C Programming

Fundamentals of C Programming

AIM:

To equip the students with basic C programming knowledge.

Fundamentals of C Programming

Objectives:

The Objectives of this module are:

- Describe the characteristics of C language
- Explain the structure of a C Program
- Describe various data types and variables in C
- Elaborate various types of operators and expressions in C
- Illustrate the functions of control statements
- Explain various types of arrays and its functions
- Explain the working of different loop statement in C

Fundamentals of C Programming

Outcome:

At the end of this module, you are expected to:

- Describe the structure of a C program
- Apply different data types in C program
- Develop a C program to show all operators
- Use appropriate control statement in a C program
- Develop a code to add two matrices using 2D array

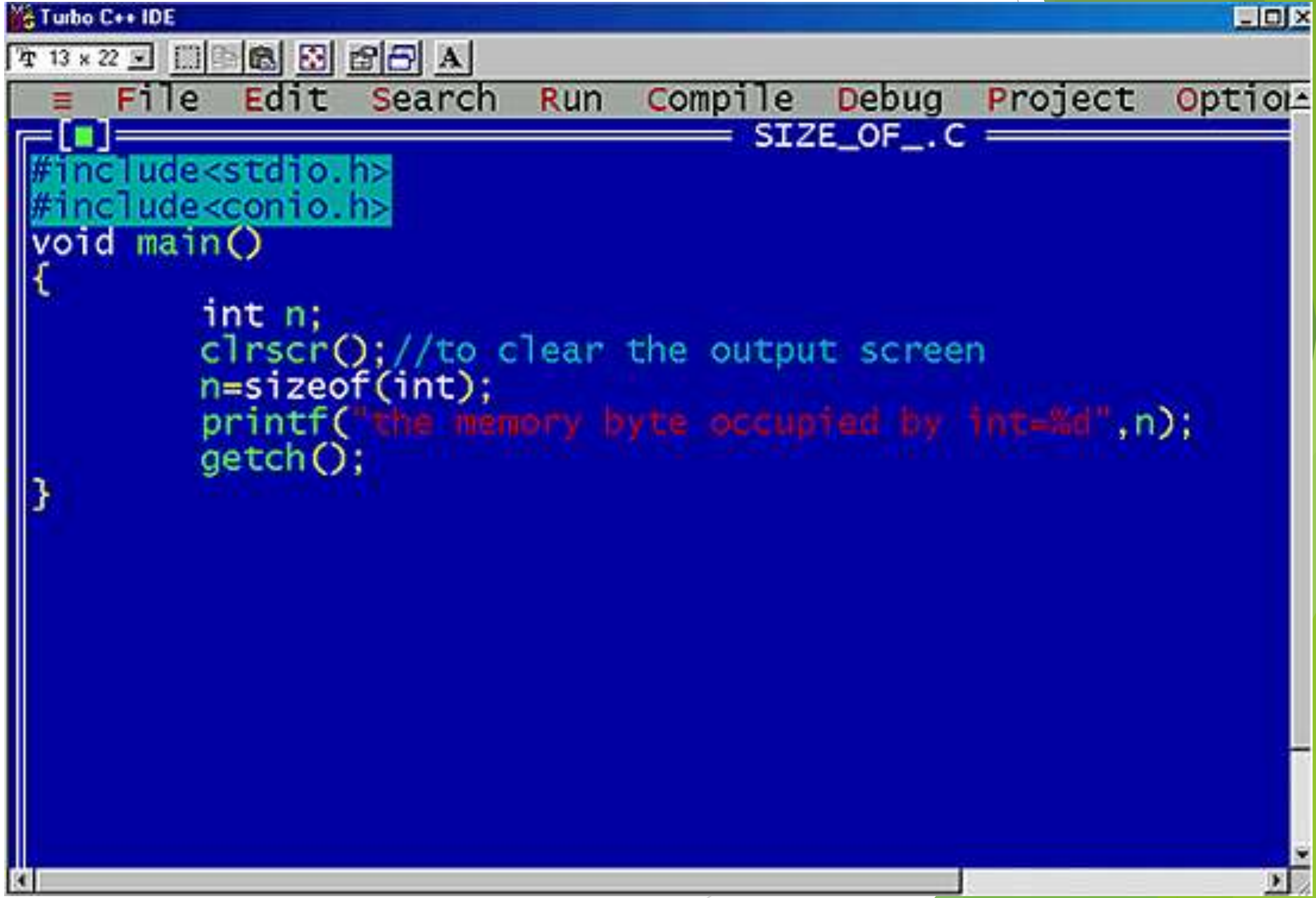
Overview of programming

Contents

1. Overview of C programming
2. Data Types
3. Constants & Variables
4. Operators & Expressions
5. Type Conversion
6. Control Statements
7. Loops

Overview of C programming

Fundamentals of C Programming



```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        clrscr();//to clear the output screen
        n=sizeof(int);
        printf("the memory byte occupied by int=%d",n);
        getch();
}
```

Fundamentals of C Programming

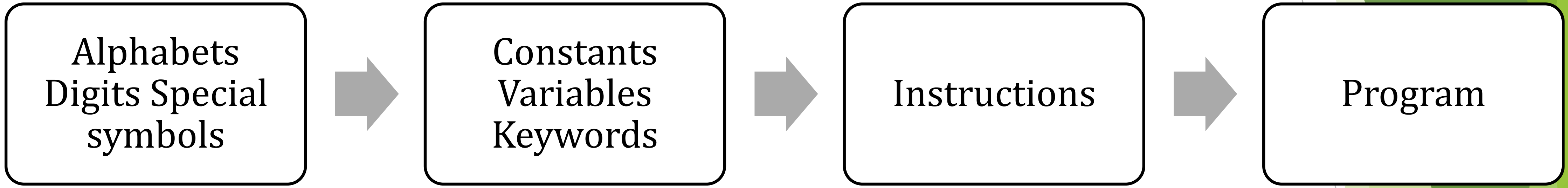
What is C programming?

C is a high-level and general-purpose programming language that is ideal for developing firmware or portable applications. Originally intended for writing system software, C was developed at Bell Labs by Dennis Ritchie for the Unix Operating System in the early 1970s.

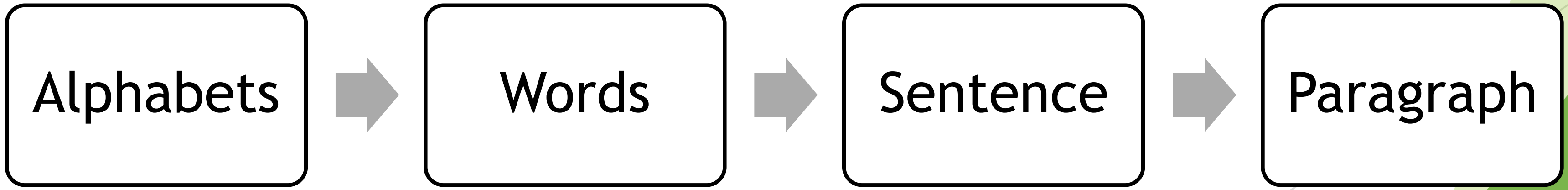
Ranked among the most widely used languages, C has a compiler for most computer systems and has influenced many popular languages – notably C++.

Fundamentals of C Programming

- *Steps in Learning C programming*


Alphabets Digits Special symbols
Constants Variables Keywords
Instructions
Program

- ***Example*** *: Steps in Learning English*



Fundamentals of C Programming

- *Steps in Learning C programming*

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{Alphabets}\\ \text{Digits Special}\\ \text{symbols}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Constants}\\ \text{Variables}\\ \text{Keywords}\end{array}} \arrow[r] & \boxed{\text{Instructions}} \arrow[r] & \boxed{\text{Program}}
\end{tikzcd}$$

• **Alphabets**	A, B, ….., Y, Z
	a, b, ……, y, z
• **Digits**	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
• **Special symbols**	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { }
	[ ] : ; " ' < > , . ? /

Fundamentals of C Programming

- ***Features of C programming***

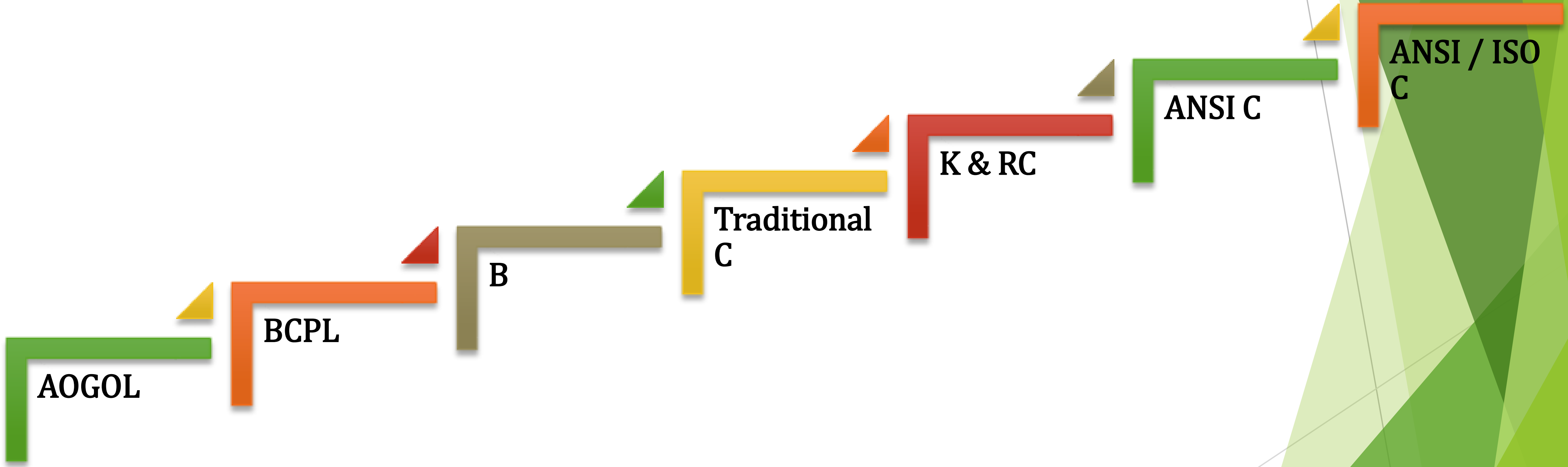
C belongs to the structured, procedural paradigms of languages. It is proven, flexible and powerful and may be used for a variety of different applications. Although high level, C and assembly language share many of the same attributes.

Some of C's most important features include:

- Fixed number of keywords, including a set of control primitives, such as if, for, while, switch and do a while.
- Multiple logical and mathematical operators, including bit manipulators.
- Multiple assignments may be applied in a single statement.
- Function return values are not always required and may be ignored if unneeded.
- Typing is static. All data has type but may be implicitly converted.
- Basic form of modularity, as files may be separately compiled and linked.
- Control of function and object visibility to other files via extern and static attributes.

Fundamentals of C Programming

- ***History of C programming***



Fundamentals of C Programming

- ***Basics of C programming***

- It is machine independent and a highly portable language.
- It has only 32 keywords or instruction set, though actual implementations include extensive library functions which enhance the basic instructions. Even a user or a programmer can write their own library functions which will increase the features and capabilities of the language.
- C language gives the provision for manipulating bits, bytes and addresses (low level activities).
- It has a large library of inbuilt functions.
- A programmer can write several building blocks known as functions within a single program. Each function performs individual tasks according to the requirement of the program.
- C compiler is easily available for all sizes of computers. Compilers are usually compact and generate object programs that are small and highly efficient when compared to programs compiled another high-level languages.
- It supports pointer implementation (Detailed discussion on pointer will be provided in

Data Types

Fundamentals of C Programming

- ***What are Data Types?***

In the C programming language, data types are declarations for memory locations or variables that determine the characteristics of the data that may be stored and the methods (operations) of processing that are permitted involving them.

Fundamentals of C Programming

- ***Data Types in C programming***

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.

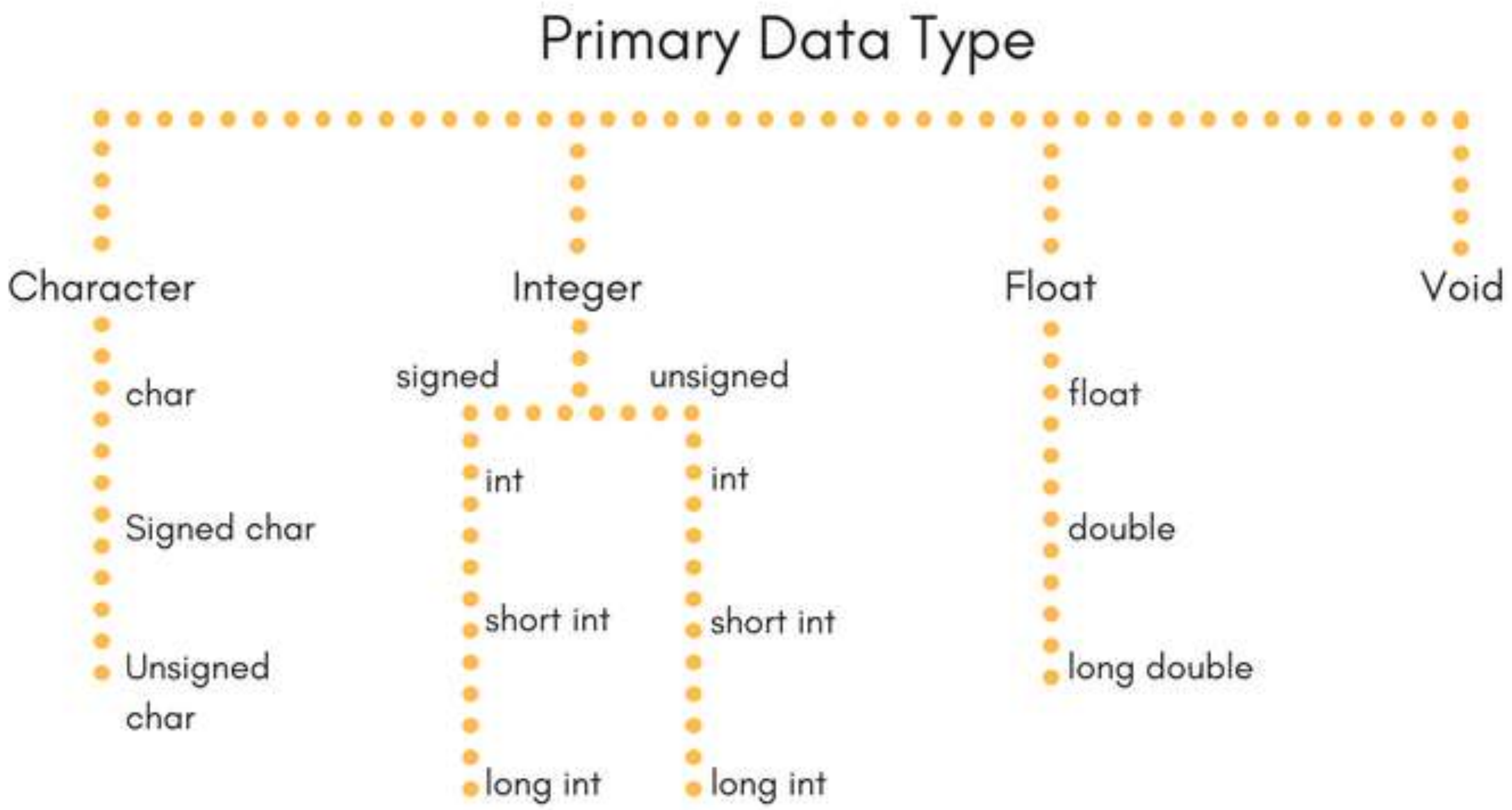
C language supports **2 different type** of data types:

Primary data types: These are fundamental data types in C namely **integer(int**), floating point(**float**), character(**char**) and **void**.

Derived data types: Derived data types are nothing but primary datatypes but a little twisted or grouped together like array, structure, union and pointer. These are discussed in details later.

Data type determines the type of data a variable will hold. If a variable x is declared as int. it means x can hold only integer values. Every variable which is used in the program must be declared as what data-type it is.

Fundamentals of C Programming



Fundamentals of C Programming

There are only a few **basic data types** in C:

char	a single byte, capable of holding one character in the local character set.
int	an integer, typically reflecting the natural size of integers on the host machine.
float	single-precision floating point.
Double	double-precision floating point.

In addition, there are a number of qualifiers that can be applied to these basic types. short and long apply to integers:

short int sh;

long int counter

Fundamentals of C Programming

- ***Data Types***
- The word **int** can be omitted in such declarations, and typically is. The intent is that short and long should provide different lengths of integers where practical; int will normally be the natural size for a particular machine. short is often **16 bits**, long **32 bits**, and int either **16 or 32 bits**.
- Each compiler is free to choose appropriate sizes for its own hardware, subject only to the restriction that shorts and ints are at least 16 bits, longs are at least 32 bits, and short is no longer than int, which is no longer than long. The qualifier signed or unsigned may be applied to char or any integer. unsigned numbers are always positive or zero, and obey the laws of arithmetic modulo 211, where n is the number of bits in the type. So, for instance, if chars are 8 bits, unsigned char variables have values between 0 and 255, while signed chars have values between -128 and 127 (in a two's complement machine). Whether plain chars are signed or unsigned is machine-dependent, but printable characters are always positive.
- The type long double specifies extended-precision floating point. As with integers, the sizes of floating-point objects are implementation-defined; **float,double** and long double could represent one, two or three distinct sizes. The standard headers **<limits. h>** and **<float. h>** contain symbolic constants for all of these sizes, along with other properties of the machine and compiler.

Constants & Variables

Fundamentals of C Programming

- ***What are Constants?***

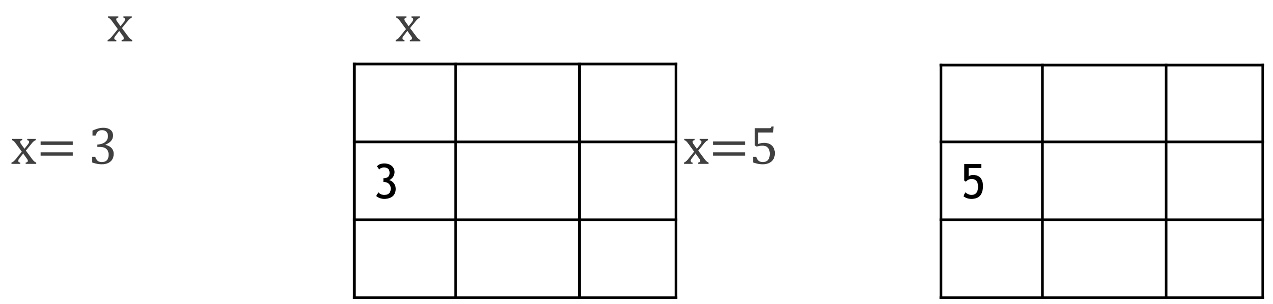
The alphabets, numbers and special symbols when properly combined form constants, variables and keywords. Let us see what are ‘constants’ and ‘variables’ in C. A constant is an entity that doesn’t change whereas a variable is an entity that may change.

In any program we typically do lots of calculations. The results of these calculations are stored in computers memory. Like human memory the computer memory also consists of millions of cells. The calculated values are stored in these memory cells. To make the retrieval and usage of these values easy these memory cells (also called memory locations) are given names. Since the value stored in each location may change the names given to these locations are called variable names.

Fundamentals of C Programming

- *Example for Constants*

Here 3 is stored in a memory location and a name x is given to it. Then we are assigning a new value 5 to the same memory location x. This would overwrite the earlier value 3, since a memory location can hold only one value at a time.



Since the location whose name is x can hold different values at different times x is known as a variable. As against this, 3 or 5 do not change, hence are known as constants.

Fundamentals of C Programming

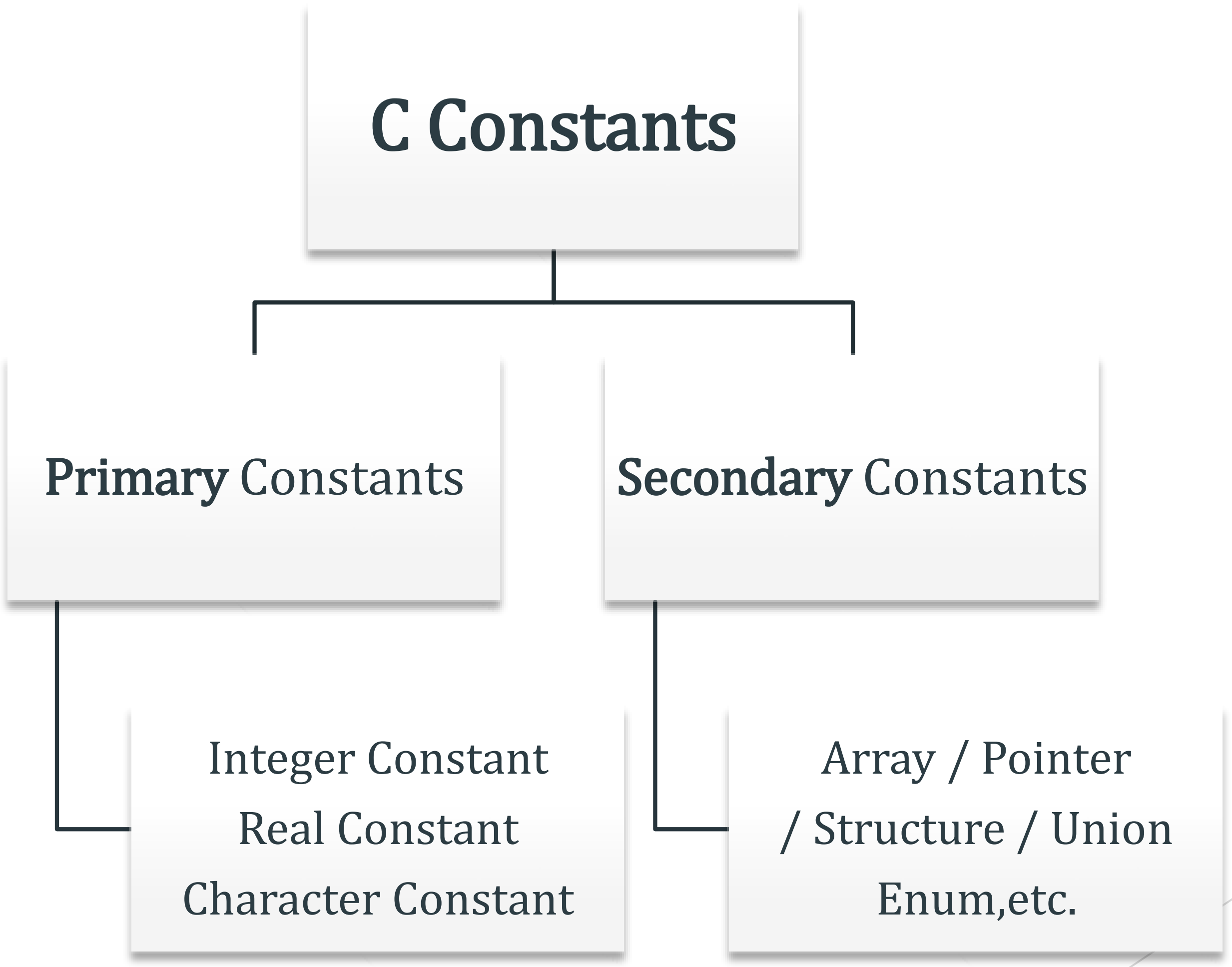
- **Types of C Constants**

C constants can be divided into two major categories:

a. Primary Constants

b. Secondary Constants

Fundamentals of C Programming



Fundamentals of C Programming

- ***Rules for Constructing Integer Constants***

  a. An integer constant must have at least one digit.

  b. It must not have a decimal point.

  c. It can be either positive or negative.

  d. If no sign precedes an integer constant it is assumed to be positive.

  e. No commas or blanks are allowed within an integer constant.

  f. The allowable range for integer constants is -32768 to 32767.

  For a 16-bit compiler like Turbo C or Turbo C++ the range is –32768 to 32767. For a 32-bit compiler the range would be even greater.

Fundamentals of C Programming

- ***Rules for Constructing Real Constants***

  Real constants are often called Floating Point constants. The real constants could be written in two forms:

  - Fractional form
  - Exponential form

  Following rules must be observed while constructing real constants expressed in **fractional form**: It must not have a decimal point.

  a. A real constant must have at least one digit.
  b. It must have a decimal point.
  c. It could be either positive or negative.
  d. Default sign is positive.
  e. No commas or blanks are allowed within a real constant.

Fundamentals of C Programming

- ***Rules for Constructing Real Constants***

Following rules must be observed while constructing real constants expressed in **exponential** form:

- The mantissa part and the exponential part should be separated by a letter e.
- The mantissa part may have a positive or negative sign.
- Default sign of mantissa part is positive.
- The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.
- Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Fundamentals of C Programming

- ***Rules for Constructing Character Constants***

a. A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to the left. For example, ’A’ is a valid character constant whereas ‘A’ is not.

b. The maximum length of a character constant can be 1 character.

Ex.: 'A'

'I'

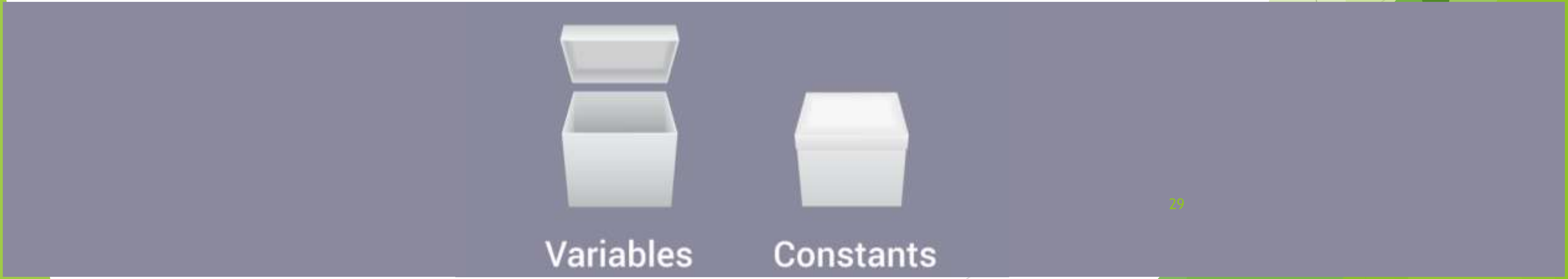
'5'

'='

Fundamentals of C Programming

- ***What are Variables?***

C Programming/**Variables**. ... **Variables** are simply names used to refer to some location in memory – a location that holds a value with which we are working. It may help to think of **variables** as a placeholder for a value. You can think of a **variable** as being equivalent to its assigned value.



Fundamentals of C Programming

- ***Types of Variables***

The things that are changing in an experiment are called variables. A variable is any factor, trait, or condition that can exist in differing amounts or types. An experiment usually has three kinds of variables:

- Independent
- Dependent
- Controlled

Fundamentals of C Programming

- **Types of Variables**

- As we saw earlier, an entity that may vary during program execution is called a variable. Variable names are names given to locations in memory. These locations can contain integer, real or character constants. In any language, the types of variables that it can support depend on the types of constants that it can handle. This is because a particular type of variable can hold only the same type of constant. For example, an integer variable can hold only an integer constant, a real variable can hold only a real constant and a character variable can hold only a character constant.
- The rules for constructing different types of constants are different. However, for constructing variable names of all types the same set of rules apply. These rules are given below.

Fundamentals of C Programming

- *Rules for Constructing Variable Names*

  - A variable name is any combination of 1 to 31 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 247 characters. Still, it would be safer to stick to the rule of 31 characters. Do not create unnecessarily long variable names as it adds to your typing effort.
  - The first character in the variable name must be an alphabet or underscore.
  - No commas or blanks are allowed within a variable name.
  - No special symbol other than an underscore (as in **gross_sal**) can be used in a variable name.

Ex.: si_int
m_hra
pop_e_89

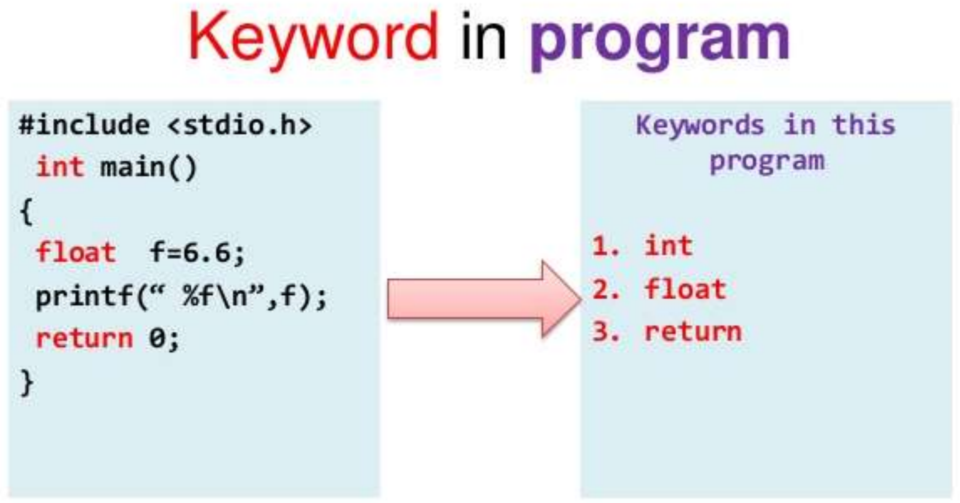
These rules remain same for all the types of primary and secondary variables.

Fundamentals of C Programming

- ***What are C Keywords?***

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. **Keywords** are part of the syntax and they cannot be used as an identifier.

For example: ... As **C** is a case sensitive language, all **keywords** must be written in lowercase.



```
#include <stdio.h>
 int main()
{
 float  f=6.6;
 printf(" %f\n",f);
 return 0;
}
```

```
1. int
2. float
3. return
```

Fundamentals of C Programming

- ***What are C Keywords?***

Keywords are the words whose meaning has already been explained to the C compiler (or in a broad sense to the computer). The keywords cannot be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer. Some C compilers allow you to construct variable names that exactly resemble the keywords. However, it would be safer not to mix up the variable names and the keywords. The keywords are also called 'Reserved words'.

There are only 32 keywords available in C. List of these keywords for your ready reference. A detailed discussion of each of these keywords would be taken up in later chapters wherever their use is relevant.

Fundamentals of C Programming

- *List of C Keywords*

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

Operators & Expressions

Fundamentals of C Programming

- ***What are Operators?***
- C language supports a rich set of built-in operators.
- An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation.
- Operators are used in programs to manipulate data and variables.

Fundamentals of C Programming

- ***Types of C Operators***

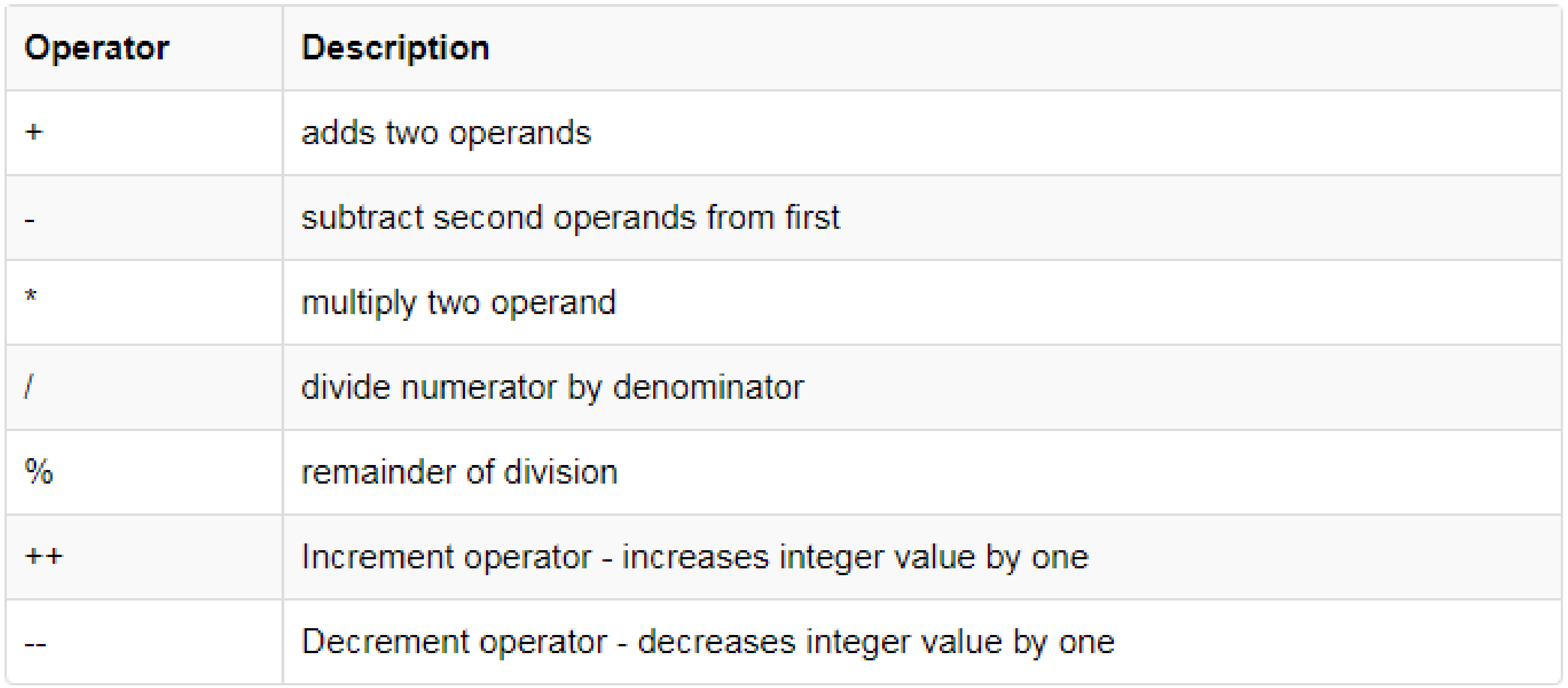
C operators can be classified into following types:

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

Fundamentals of C Programming

- ***Arithmetic Operators***

C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators



Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

Fundamentals of C Programming

- ***Example for Arithmetic Operators***

  For example, a year is a leap year if it is divisible by 4 but not by 100, except that years divisible by 400 are leap years. Therefore

  if «year" 4 == 0 && year" 100 1= 0) :: year % 400 == 0)

  printf(lI"d is a leap year\nllt year);

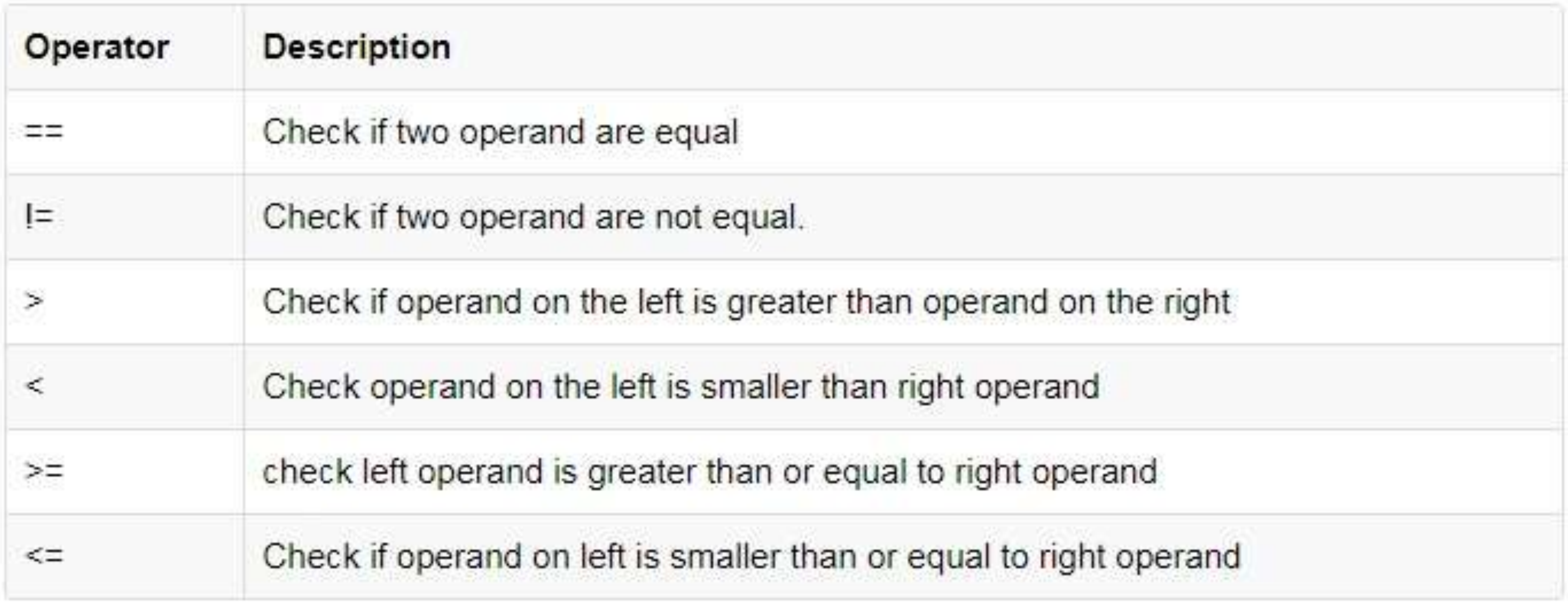
  else

  printf (11%<1 is not a leap year\n II t year);

Fundamentals of C Programming

- ***Relational Operators***

The following table shows all relation operators supported by C.

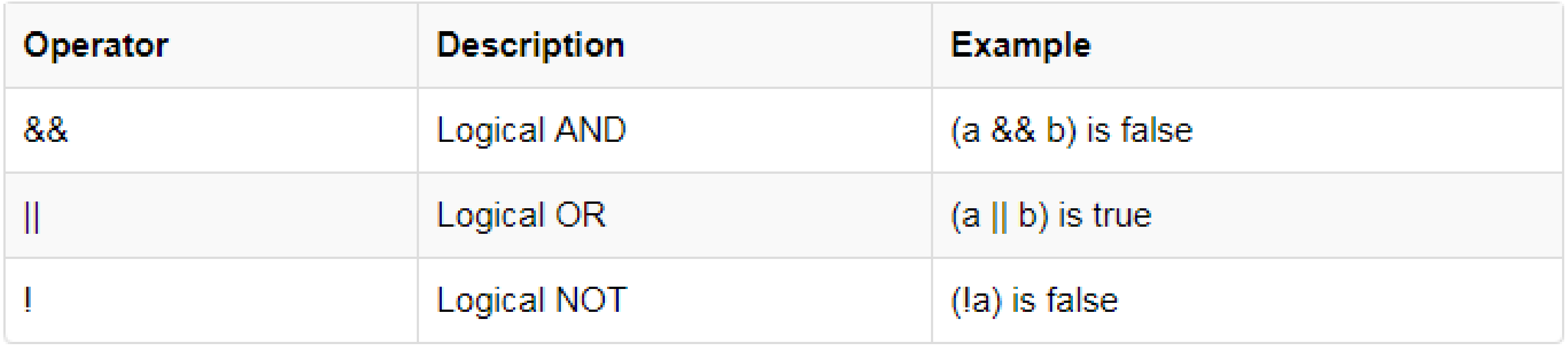


Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

Fundamentals of C Programming

- ***Logical Operators***

C language supports following 3 logical operators. Suppose a = 1 and b = 0,



Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Fundamentals of C Programming

- ***Example for Relational and logical Operators***

For example, here is a loop from the input function get line that we wrote in,

Chapter 1:

for (i=0; i<lim-1 && (c=getchar(» 1= '\n' && c 1= EOF; ++i)

s[i] = c;

Before reading a new character it is necessary to check that there is room to store it in the array s, so the test i < lim-1 must be made first. Moreover, if this test fails, we must not go on and read another character.

Fundamentals of C Programming

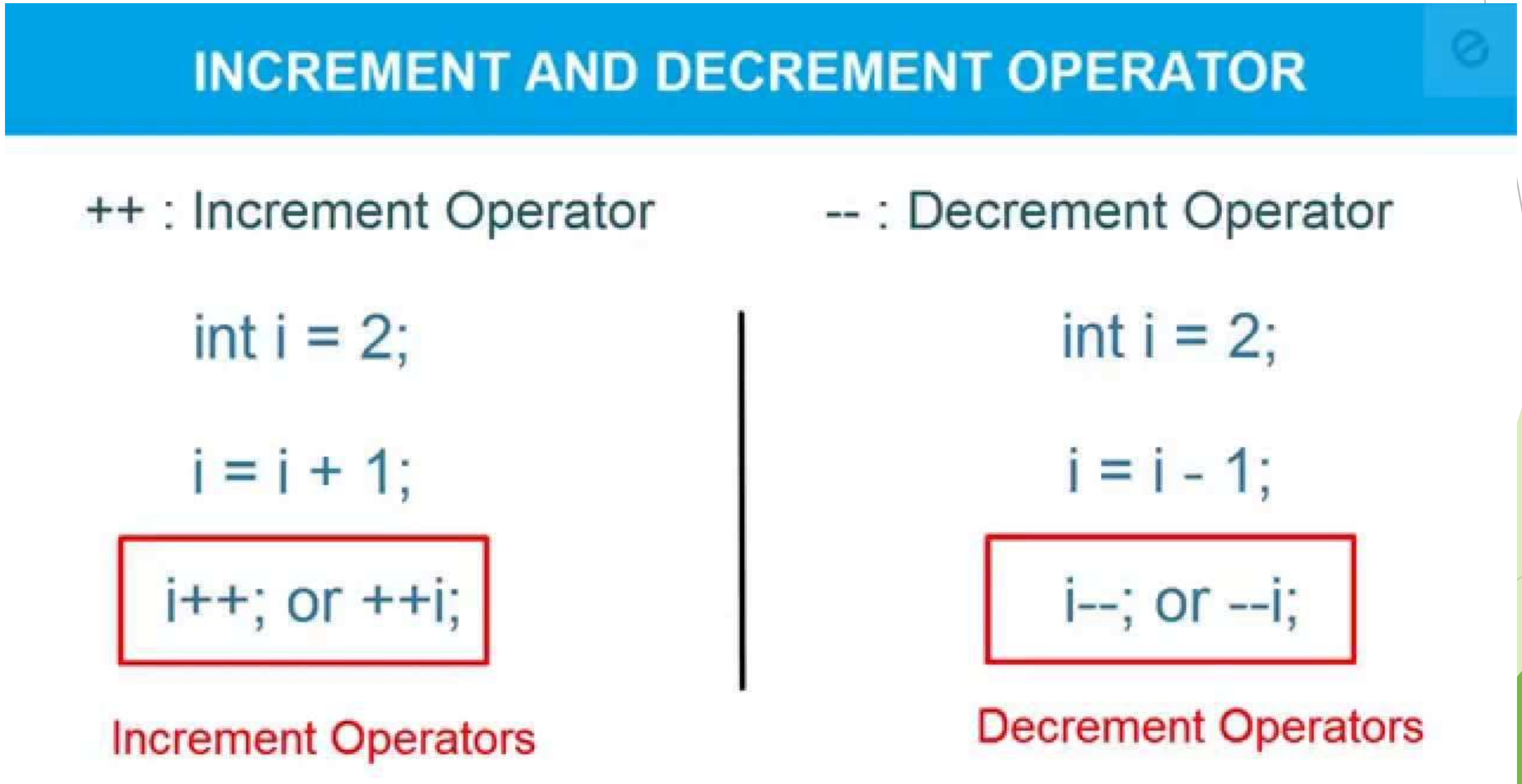
- ***Increment and Decrement Operators***

  Increment and decrement operators are unary operators that add or subtract one from their operand, respectively. They are commonly implemented in imperative programming languages. C-like languages feature two versions (pre- and post-) of each operator with slightly different semantics.

  - The **pre-increment** and **pre-decrement** operators increment (or decrement) their operand by 1, and the value of the expression is the resulting incremented (or decremented) value.
  - The **post-increment** and **post-decrement** operators increase (or decrease) the value of their operand by 1, but the value of the expression is the operand's original value prior to the increment (or decrement) operation

Fundamentals of C Programming

- *Increment and Decrement Operators*



INCREMENT AND DECREMENT OPERATOR

++ : Increment Operator	-- : Decrement Operator
int i = 2;	int i = 2;
i = i + 1;	i = i - 1;
i++; or ++i;	i--; or --i;
Increment Operators	Decrement Operators

Fundamentals of C Programming

- ***Examples for Increment and Decrement Operators***

The following C code fragment illustrates the difference between the *pre* and *post* increment and decrement operators:

```
int x; int y;
 // Increment operators
x = 1;     // x is now 2, y is also 2 y = x++;
 y = ++x;  // x is now 3, y is 2
// Decrement operators
x = 3;
y = x--;  // x is now 2, y is 3
y = --x; // x is now 1, y is also 1
```

Fundamentals of C Programming

- ***Examples for Increment and Decrement Operators***

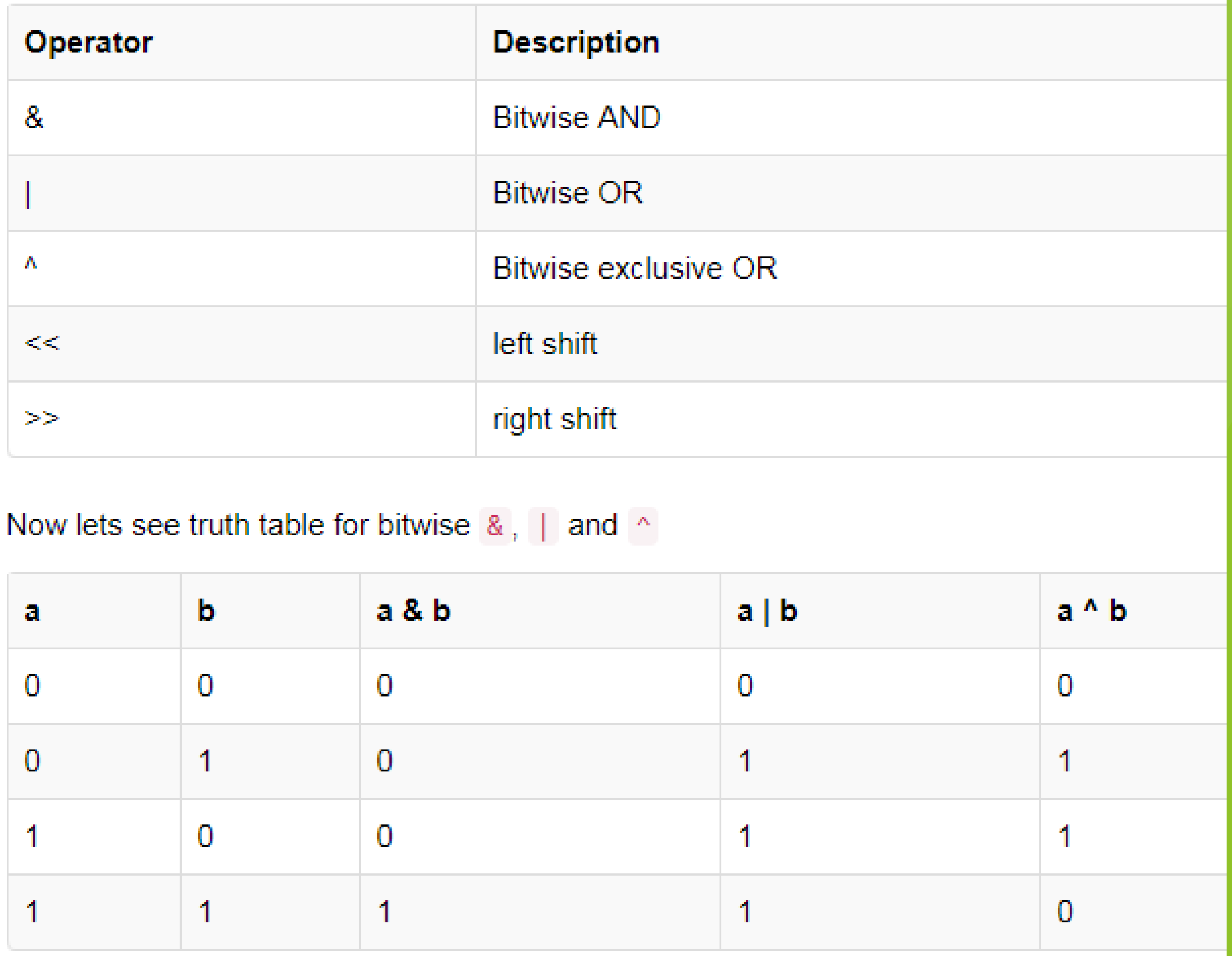
The post-increment operator is commonly used with array subscripts. For example:

```
// Sum the elements of an array
float sum_elements(float arr[], int n)
{
   float  sum = 0.0;
   int    i =   0;
   while (i < n)
      sum += arr[i++];    // Post-increment of i, which steps
                          //  through n elements of the array
   return sum;
}
```

Fundamentals of C Programming

- ***Bitwise Operators***

Bitwise operators perform manipulations of data at bit level. These operators also perform shifting of bits from right to left. Bitwise operators are not applied to float or double(These are datatypes, we will learn about them in the next tutorial).



Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

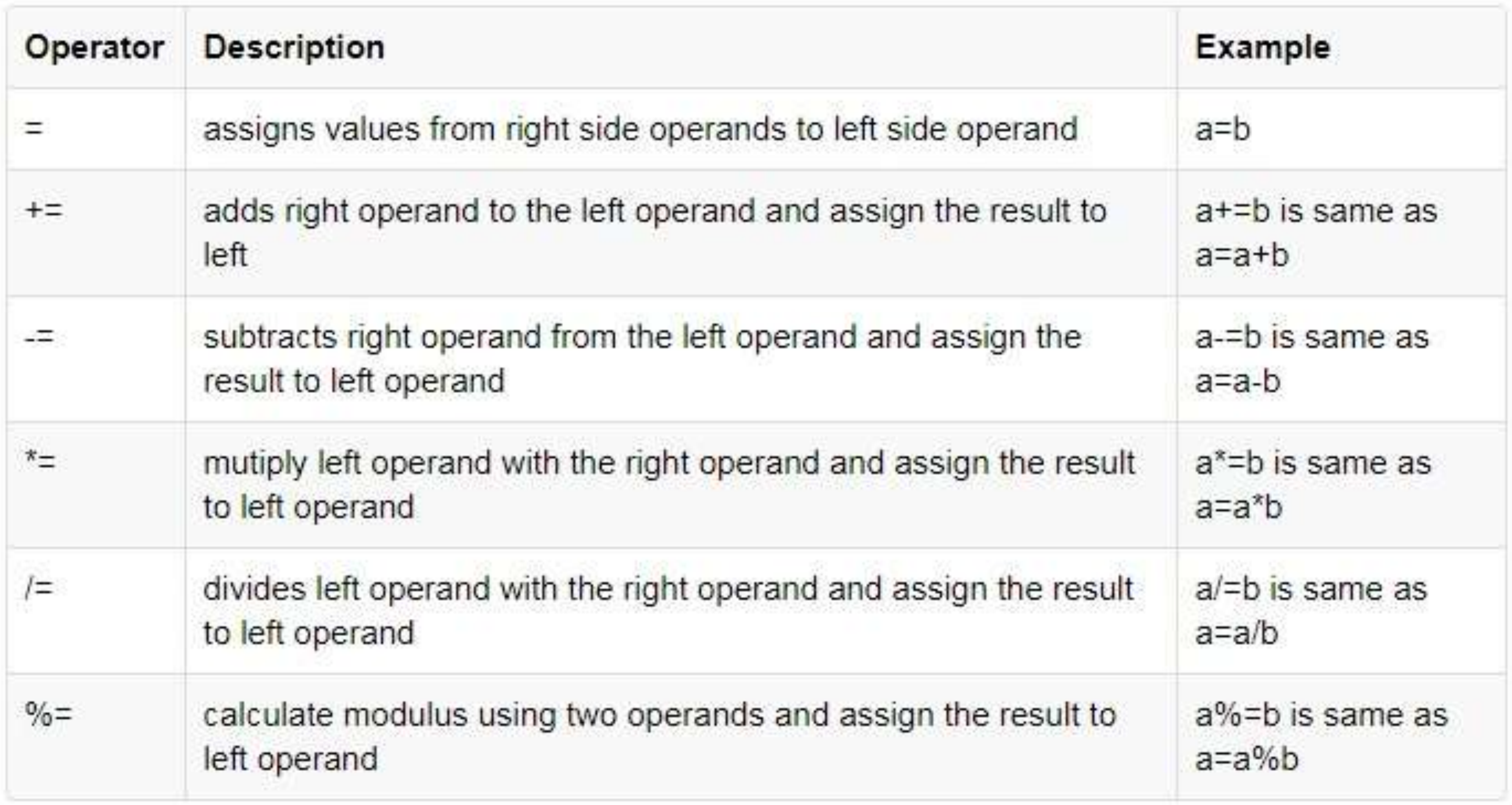
Now lets see truth table for bitwise `&`, `|` and `^`

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Fundamentals of C Programming

- ***Assignment Operators***

  Assignment operators supported by C language are as follows:



Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

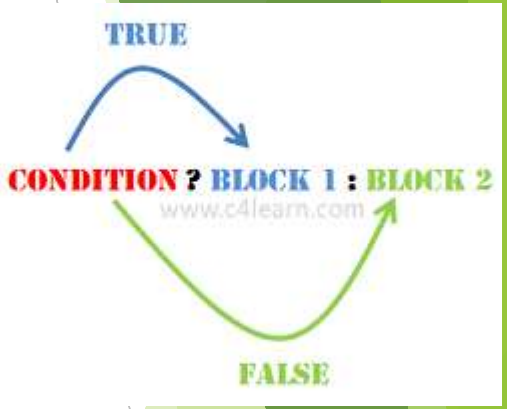
Fundamentals of C Programming

- ***Conditional Operators***

The conditional operators in C language are known by two more nam

- **Ternary Operator**
- **? : Operator**

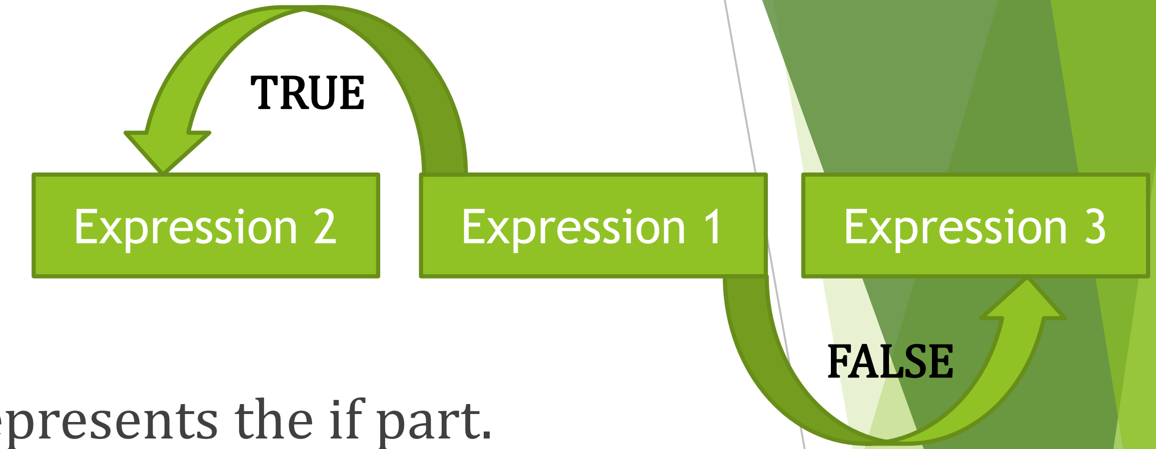
It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.



expression 1 expression 2: expression 3

Fundamentals of C Programming

- ***Conditional Operators***

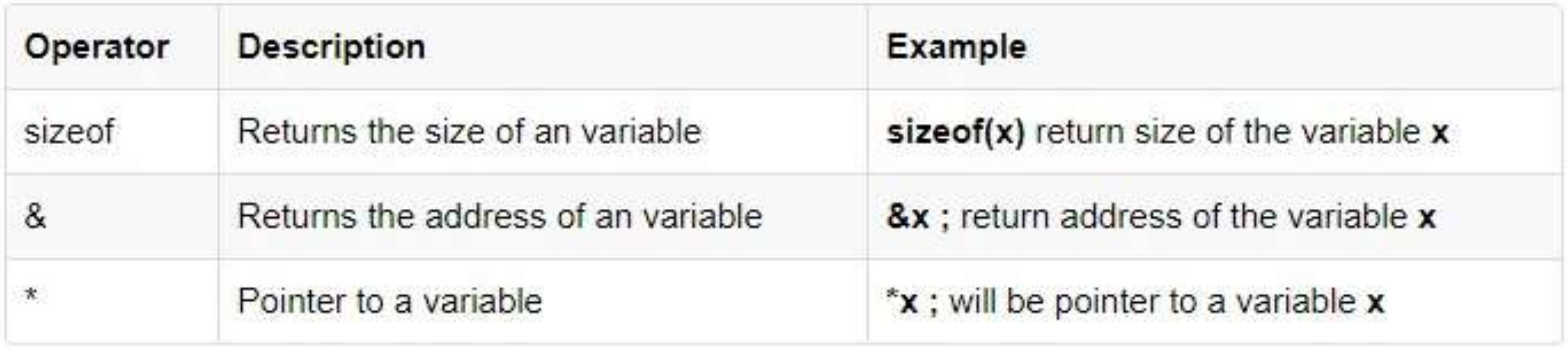


Explanation:

- The question mark "?" in the syntax represents the if part.
- The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3).
- If (expression 1) returns true then the expression on the left side of " : " i.e., m (expression 2) is executed.
- If (expression 1) returns false then the expression on the right side of " : " i.e (expression 3) is executed.

Fundamentals of C Programming

- *Special Operators*



Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x** ; return address of the variable **x**
*	Pointer to a variable	***x** ; will be pointer to a variable **x**

Fundamentals of C Programming

- ***BODMAS Rule:***

  Brackets of Division Multiplication Addition Subtraction Brackets will have the highest precedence and have to be evaluated first, then comes of , then comes division, multiplication, addition and finally subtraction. C language uses some rules in evaluating the expressions and they r called as precedence rules or sometimes also referred to as hierarchy of operations, with some operators with highest precedence and some with least.

  The 2 distinct priority levels of arithmetic operators in c are-

  Highest priority: * / %

  Lowest priority: + - Precedence of operators

Fundamentals of C Programming

- ***Expression***
  - The combination of operators and operands is said to be an expression.
  - An arithmetic expression is a combination of variables, constants, and operators .
  - arranged as per the syntax of the language.
  - E.g: 1) a = x + y

Fundamentals of C Programming

In programming, an expression is any legal combination of symbols that represents a value. Each programming language and application has its own rules for what is legal and illegal. For **example**, in the C language x+5 is an expression, as is the character string "MONKEYS."

Every expression consists of at least one operand and can have one or more operators. Operands are values, whereas operators are symbols that represent particular actions. In the expression:

x + 5

x + 5

x and 5 are operands, and + is an operator.

Expressions are used in programming languages, database systems, and spreadsheet applications. For example, in database systems, you use expressions to specify which information you want to see. These types of expressions are called queries.

Fundamentals of C Programming

- ***Examples for Expression***

Expressions are often classified by the type of value that they represent. For example:

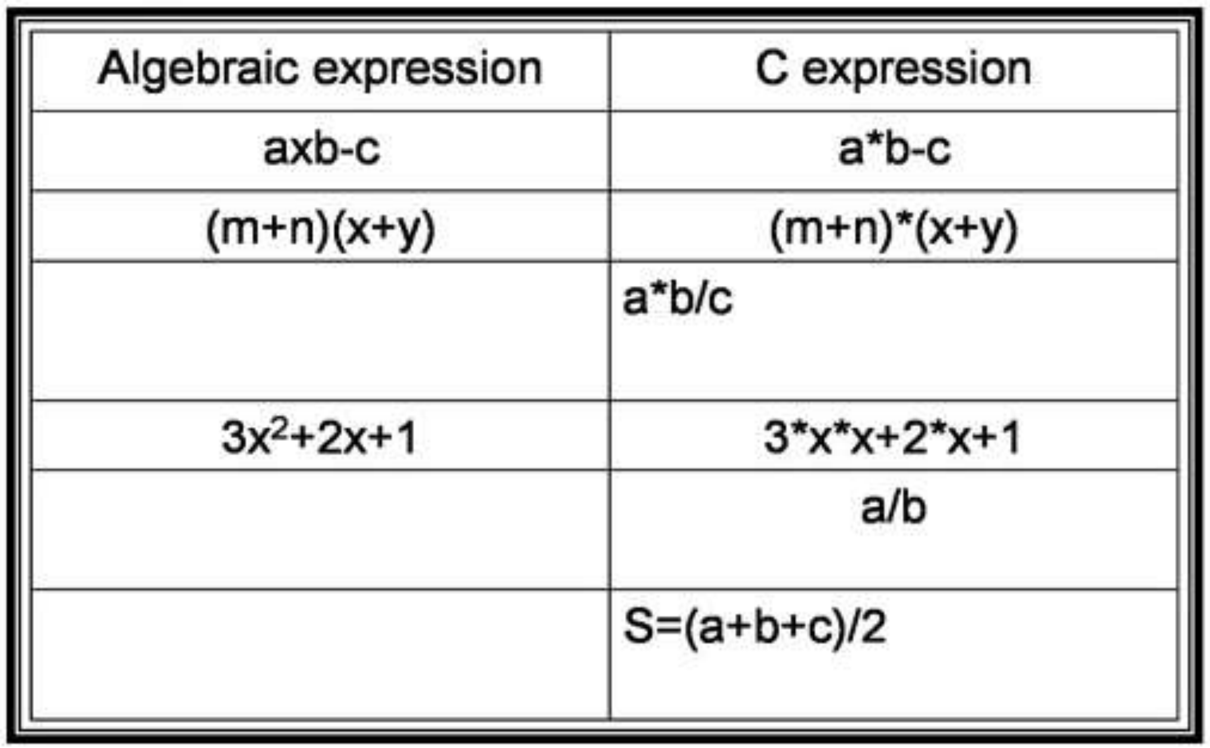
- Boolean expressions : Evaluate to either TRUE or FALSE
- integer expressions: Evaluate to whole numbers, like 3 or 100
- Floating-point expressions: Evaluate to real numbers, like 3.141 or -0.005
- String expressions: Evaluate to character strings

Fundamentals of C Programming

- ***Arithmetic Expression***

An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

Eg 1) a = x + y;



Algebraic expression	C expression
axb-c	a*b-c
(m+n)(x+y)	(m+n)*(x+y)
	a*b/c
$3x^2+2x+1$	3*x*x+2*x+1
	a/b
	S=(a+b+c)/2

Fundamentals of C Programming

- ***Evaluation of Expression***
  - Expressions are evaluated using an assignment statement of the form variable = expression.
  - Eg: 1) x = a * b – c; 2) y = b / c * a;
  - An arithmetic expression without parenthesis will be evaluated from left to right using the rule of precedence of operators.

There are two distinct priority levels of arithmetic operators in C.

- High priority * / %
- Low priority + -

Fundamentals of C Programming

- ***Rules for Evaluation of Expression***
  - First parenthesized sub expression from left to right are evaluated.
  - If parentheses are nested, the evaluation begins with the innermost sub expression.
  - The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
  - The associatively rule is applied when 2 or more operators of the same precedence level appear in a sub expression.
  - Arithmetic expressions are evaluated from left to right using the rules of precedence.
  - When parentheses are used, the expressions within parentheses assume highest priority.

Type Conversions

Fundamentals of C Programming

- ***Difference between Typecasting and Type Conversions***

Typecasting, or **type conversion**, is a method of changing an entity from one data type to an

BASIS FOR COMPARISON	TYPE CASTING	TYPE CONVERSION
Definition	When a user can convert the one data type into other then it is called as the type casting.	Type Conversion is that which automatically converts the one data type into another.
Implemented	Implemented on two 'incompatible' data types.	Implemented only when two data types are 'compatible'.
Operator	For casting a data type to another, a casting operator '()' is required.	No operator required.
Implemented	It is done during program designing.	It is done explicitly while compiling.
Conversion type	Narrowing conversion.	Widening conversion.

Example	int x;
	byte y;
	…
	…
	y= (byte) x;
		int x=3;
		float y;
		y=a; // value in y=3.000.

Fundamentals of C Programming

- ***Types Conversions***

The compiler will automatically change one type of data into another if it makes sense. For instance, if you assign an integer value to a floating-point variable, the compiler will convert the int to a float. Casting allows you to make this type conversion explicit, or to force it when it wouldn't normally happen.

There are two type of conversions in C:

- **Implicit** type conversion
- **Explicit** type conversion

Fundamentals of C Programming

- **Implicit type conversions**

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 lose 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

Fundamentals of C Programming

- ***Example of Implicit type conversions***

```
// 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'
   x = x + y; is 97
   // 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
```

Fundamentals of C Programming

- ***Explicit type conversions***

  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.

Fundamentals of C Programming

- ***Example of Explicit type conversions***

```
// C program to demonstrate explicit type casting
#include<studio.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
```

Fundamentals of C Programming

- ***Advantages of type conversions***

  - 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.

Control Statements

Fundamentals of C Programming

- ***Definition for control statements***

  Control statements enable us to specify the flow of program control; i.e., the order in which the instructions in a program must be executed. They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

  - If Statement
  - If / else statement
  - For loop
  - Do loop
  - Do While Loop

Fundamentals of C Programming

➢ ***Control statements***

If Statement	If statement is a conditional branching statement. In conditional branching statement a condition is evaluated, if it is evaluate true a group of statement is executed.
Switch Statement	Switch statements simulate the use of multiple if statement. The switch statement is probably the single most syntactically awkward and error-prone feature of the C language.
Goto Statement	The goto statement is used to alter the normal sequence of program execution by transferring control to some other part of the program unconditionally.
For loop	For loop in C is the most general looping construct. The loop header contains three parts: an initialization, a continuation condition, and step.
While loop	The while statement is also a looping statement. The while loop evaluates the test expression before every loop, so it can execute zero times if the condition is initially false.
Do-While loop	Is also used for looping. In this case the loop condition is tested at the end of the body of the loop. Hence the loop is executed at least one.
Break Statement	The break statement is a jump instruction and can be used inside a switch construct, for loop, while loop and do-while loop.
Continue Statement	Continue statement is a jump statement. The continue statement can be used only inside for loop, while loop and do-while loop.
Nested Loop	In many cases we may use loop statement inside another looping statement. This type of looping is called nested loop.

Fundamentals of C Programming

- ***Types of control statements***

Control statements are been classified in following types:

- Decision making statements: ***if , if else and nested if*** .
- Selection statements: ***if and switch*** .
- Iteration statements: ***while, do, and for*** .
- Jump statements: ***break, continue, return*** *and* ***goto*** .

Other C statements:

– Compound statement

– Null statement

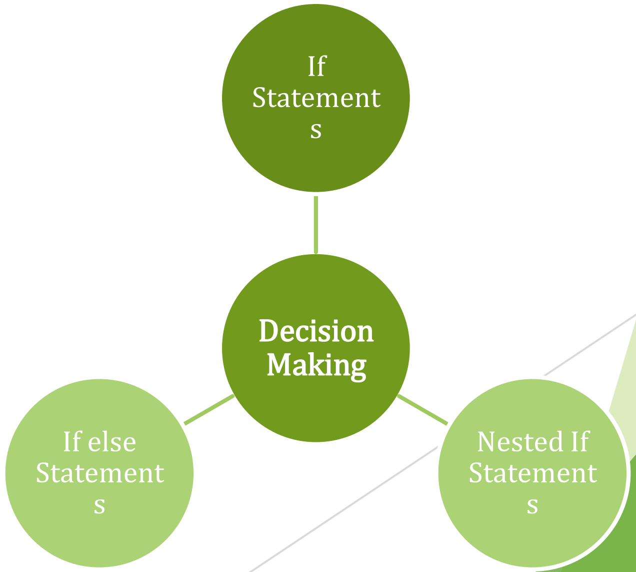
Fundamentals of C Programming

- ***Decision Making statements***

In decision control statements (if-else and nested if), group of statements are executed when condition is true. If condition is false, then else part statements are executed.

There are 3 types of decision making control statements in C language. They are:

- if statements
- if else statements
- nested if statements



Fundamentals of C Programming

- ***"IF", "ELSE" AND "NESTED IF" DECISION CONTROL STATEMENTS IN C:***

Decision control statements	**Syntax/Description**
if	Syntax: if (condition) { Statements; } Description: In these type of statements, if condition is true, then respective block of code is executed.

if…else	Syntax:
	if (condition)
	{ Statement1; Statement2; }
	else
	{ Statement3; Statement4; }Description:
	In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

nested if	Syntax:
	if (condition1){ Statement1; }
	else if (condition2)
	{ Statement2; }
	else Statement 3;Description:
	If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed.

Fundamentals of C Programming

- ***IF statements***
- If statement is a conditional branching statement.
- In conditional branching statement a condition is evaluated, if it is evaluate true a group of statement is executed. The simple format of an if statement is as follows:

if (expression)

statement

- If the expression is evaluated and found to be true, the single statement following the "if statement" is executed. If false, the following statement is skipped. Here a compound statement composed of several statements bounded by braces can replace the single statement.

Fundamentals of C Programming

- ***Example for IF statements***

- In “if” control statement, respective block of code is executed when condition is true.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int  number;
clrscr();
printf(“Enter a number \n”);
scanf(“%d”&number);
if(number>0)
printf(“Given number is positive \n”);
getch();
return 0;
}
```

Output:

```
Enter a number
  7
  Given number is positive
```

Fundamentals of C Programming

- ***IF Else statements***
  - This feature permits the programmer to write a single comparison, and then execute one of the two statements depending upon whether the test expression is true or false. The general form of the if-else statement is:

```
if(expression)
statement1
else
statement2
```

Here also expression in parentheses must evaluate to (a boolean) true or false. Typically you’re testing something to see if it’s true, and then running a code block(one or more statements) if it is true, and another block of code if it isn’t. The statement1 or statement2 can be either simple or compound statement.

Fundamentals of C Programming

- ***Example for IF else statements***

- In “if” control statement, respective block of code is executed when condition is true.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int number;
clrscr();
printf(“Enter a number\n”);
scanf(“%d”,&number);
if(number==0)
printf(“Given number is Zero\n”);
else
printf(“Given number is not Zero\n”);
getch();
return 0;
}
```

Output:

```
Enter a number
    5
    Given number is not Zero
```

Fundamentals of C Programming

- ***IF Else statements for multiple conditions***

- You can set up an if-else statement to test for **multiple conditions**. The following example uses two conditions so that if the first test fails, we want to perform a second test before deciding what to do:

```
if (x%2==0)
{
printf("x is an even number");
}
else
{
if (x>10)
{
printf("x is an odd number and greater than 10");
}
else
{
printf("x is an odd number and less than 10");
}
}
```

Fundamentals of C Programming

- *IF Else statements for multiple conditions*

  This brings up the other if-else construct, the if, else if, else. This construct is useful where two or more alternatives are available for selection.

  The syntax is

```
if(condition)
statement 1;
else if (condition)
statement 2;
…………………
…………………
else if(condition)
statement n-1;
else
statement n ;
```

  The various conditions are evaluated one by one starting from top to bottom, on reaching a condition evaluating to true the statement group associated with it are executed and skip other statements. If none of expression is evaluate to true, then the statement or group of statement associated with the final else is executed.

Fundamentals of C Programming

- ***Example for IF Else statements for multiple conditions***

```
#include<stdio.h>
#include<conio.h>
void main()
{
int number;
clrscr();
printf("Enter a number\n");
scanf("%d"&number);
if(number==0)
printf("\n\n Given number is zero\n");
else if (number > 0)
printf("Given number is Positive\n");
else
printf("Given number is Negative\n");
getch();
}
```

Output:

```
    Enter a number
         96
        Given number is Positive
```

Fundamentals of C Programming

- ***Nested IF statements***

- The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

- The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.

Fundamentals of C Programming

- **Syntax of Nested IF statements**

```
if (testExpression1)
{
  // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
  // statements to be executed if testExpression1 is false and testExpression2 is true
}
else if (testExpression 3)
{
  // statements to be executed if testExpression1 and testExpression2 is false and testExpression3 is true
}
.
.
else
{
  // statements to be executed if all test expressions are false
}
```

Fundamentals of C Programming

➢ *Examples of Nested IF statements*

```
// Program to relate two integers using =, > or <
#include <stdio.h>
int main()
{
int number1, number2;
printf("Enter two integers: ");
scanf("%d %d", &number1, &number2);
//checks if two integers are equal.
if(number1 == number2) { printf("Result: %d = %d",number1,number2);
 }
//checks if number1 is greater than number2.
else if (number1 > number2)
 {
 printf("Result: %d > %d", number1, number2);
}
 // if both test expression is false
else
{
printf("Result: %d < %d",number1, number2);
}
return 0;
}
```

Output:

```
Enter two integers: 12
23
Result: 12 < 23
```

Fundamentals of C Programming

- ***Switch statement:***

  The expression that forms the argument of switch is evaluated to either char or integer. Similarly the constant expression follows the keyword case should be a value int or char. That is, names of variable can also be used. Switch can test for only equality.

  Take a look at the following if-else code, and notice how confusing it can be to have nested if tests, even just a few levels deep:

➢ Switch statement:

Switch statement simulate the use of multiple if statement. The switch statement is probably the single most syntactically awkward and error-prone feature of the C language. Syntax of c switch statement is

```
switch(expression)
{
case constant1:
statements 1;
break;
case constant2:
statements 2;
break;
……………………
default:
statements n;
break;
}
```

When the **switch statement** is executed, the expression in the switch statement is evaluated and the control is transferred directly to the group of statements whose case label value matches the value of the expression. Each group of statement ends with a break statement. The execution of break statement causes immediate exit from the switch statement. If break statement is not present, then the execution will falls trough all the statement in the selected case. If none of the case-label value matches the value of expression, then none of the group within the switch statement will be selected and the control is transferred directly to the statement that follows the switch statement c.

Fundamentals of C Programming

- ***Examples of Switch statements***

```
int number = 10;
if(number == 1)
{
printf("Given number is 1\n");
}
else if((number == 2)
{
printf("Given number is 2\n");
}
else if((number ==3)
{
printf("Given number is 3\n");
}
else if((number ==4)
{
printf("Given number is 4\n");
}
else if((number ==5)
{
printf("Given number is 5\n");
}
else
{
if(number<0)
printf("Given number is negative\n");
else
printf("Given number is greater than 5\n");
}
```

Fundamentals of C Programming

- **Example of Switch statement:**

  It’s illegal to have more than one case label using the same value. For example, the following block of code won’t compile because it uses two cases with the same.

```
int temp = 100;
switch(temp)
{
case 10 : printf(“10”);
break;
case 10 : printf(“10”); // won’t compile!
break;
case 100 :printf(“1000”);
break;
default : printf(“default”);
break;
}
```

Fundamentals of C Programming

- ***Break and Fall Through in Switch statement:***

  When case constants are evaluated from the top to down, and the first case constant that matches the switch's expression is the execution entry point. In other words, once a case constant is matched, C will execute the associated code block, and all subsequent code blocks.

Output:

```
Enter a number between 1 and 5
11

Default : Given number is 3
          Given number is 4
          Given number is 5
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
int number;
clrsr();
printf("Enter  a number  between 1 and 5\n");
scanf("%d",&number);
switch (number)
{
case 1 :
printf("Given number is 1 \n");

case 2 :
printf("Given number is 2 \n");

default :
printf("Default \n");

case 3 :
printf("Given number is 3 \n");

case 4 :
printf("Given number is 4 \n");

case 5 :
printf("Given number is 5 \n");
}
getch();
}
```

Fundamentals of C Programming

- ***Goto statement:***

The goto statement is used to alter the normal sequence of program execution by transferring control to some other part of the program unconditionally.

Syntax :

goto label:

where the label is an identifier that is used to label the target statement to which the control is transferred. Control may be transferred to anywhere within the current function. The target statement must be labeled, and a colon must follow the label. Thus the target statement will appear as

label: statement:

Each labeled statement within the function must have a unique label, i.e., no two statement can have the same label.

Fundamentals of C Programming

- *Example program for Goto statement:*

```
#include<stdio.h>
#include<conio.h>
void main()
{
int  number;
clrscr();
printf(“www.”);
goto x;
y:
printf(“mode”);
goto z;
x:
printf(“learner”);
goto y;
z:
printf(“.com”);
getch();
}
```

Output:

```
www.learnermode.com
```

Fundamentals of C Programming

- **For Loop statement:**

  For statement or loop in C is the most general looping construct. The loop header in for loop contains three parts: an initialization, a continuation condition, and step.

  Syntax:

```
for (initialization, condition, step)
{
Statement 1;
Statement 2;
……………
Statement n;
}
```

Fundamentals of C Programming

- **For Loop statement:**
  - For statement contain three parts separated by two semicolons. The first part is known as initialization. The variable called loop control variable or index variable is initialized in this part.
  - The second part is known as the condition. The condition should be a value one. The condition check can be a compound expression made up of relational expression connected by logical AND, OR.
  - The third part is known as step. It should be an arithmetic expression. The initialization need not be contained to a single variable. If more than one variable is used for initialization they are separated by commas. The step can also applied to more than one variable separated by commas.

    When for statement is encountered at first index variable is get initialized. This process is done only once during the execution of for statement. When condition is evaluated, if it is true the body of the loop will be executed. If more than one statement has to be executed it should end with a pair of braces. The condition of for loop is executed each time at the beginning of the loop. After executing the body of the loop the step is executed, again the condition is executed. If the condition become false it exit from the loop and control transferred to the statement

Fundamentals of C Programming

- **Example program For Loop statement:**

```
#include<stdio.h>
#include<conio.h>
int main()
{
int   i;
clrscr();
for(i=1;i<=5;i++)
printf("%d\n",i);
printf("\n");
}
getch();
return 0;
}
```

Output:

```
1
2
3
4
5
```

Fundamentals of C Programming

- ***While Loop statement:***

The while statement is also a looping statement. The while loop evaluates the test expression before every loop, so it can execute zero times if the condition is initially false. It has the following syntax.

```
while (expression)
{
Statement 1;
Statement 2;
...............
Statement n;
}
```

Fundamentals of C Programming

- ***While Loop statement:***

  Here the initialization of a loop control variable is generally done before the loop separately. The testing of the condition is done in while by evaluating the expression within the parenthesis. If the evaluation result is true value, the block of statement within calibrates is executed. If the evaluation result is false value the block of statements within the body of loop is skipped and the loop execution get terminated with the control passing to the statement immediately following the while construct. The increment or decrement of the loop control variable is generally done within the body of the loop.

Fundamentals of C Programming

- *Example program while Loop statement:*

```
#include<stdio.h>
#inlude<conio.h>
int  main()
{
int i;
clrscr();
i= 1;
while(i<=5)
{
printf(“%d\n”,i);
i++;
}
getch();
return 0;
}
```

Output:

```
1
2
3
4
5
```

Fundamentals of C Programming

- ***Do - While Loop statement:***

This construct is also used for looping. In this case the loop condition is tested at the end of the body of the loop. Hence the loop is executed at least one. The do-while loop is an unpopular area of the language, most programmers' tries to use the straight while if it is possible.

Syntax of do while loop is

```
do
{
Statement 1;
Statement 2;
...............
Statement n;
}
while(expression);
```

Fundamentals of C Programming

- ***Do - While Loop statement:***

  Here the block of statement following the do is executed without any condition check. After this expression is evaluated and if it is true the block of statement in the body of the loop is executed again. Thus the block of statement is repeatedly executed till the expression is evaluated to false.

  do-while construct is not used as often as the while loops or for loops in normal case of iteration but there are situation where a loop is to be executed at least one, in such cases this construction is very useful.

Fundamentals of C Programming

- *Example for Do - While Loop statement:*

```
#include<stdio.h>
#include<conio.h>
int main()
{
int  i;
clrscr();
i=1;
do
{
printf("%d\n",i);
i++;
}
while(i<5);
getch();
return 0;
}
```

Output:

```
1
2
3
4
5
```

Fundamentals of C Programming

- ***Break statement:***
  - The break statement is a jump instruction and can be used inside a switch construct, for loop, while loop and do-while loop.
  - The execution of break statement causes immediate exit from the concern construct and the control is transferred to the statement following the loop.
  - In the loop construct the execution of break statement terminates loop and further execution of the program is reserved with the statement following the body of the loop.

Fundamentals of C Programming

➢ ***Example for Break statement:***

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i;
clrscr();
for(i=1;i<10;i=i+1)
{
if(i==2)
break;
printf(“%d\n”,i);
}
While(i<=10)
{
if(i==4)
break;
printf(“%d\n”,i);
i=i+1;
}
do
{
if(i==6)
break;
printf(“%d\n”,i);
i=i+1;
}
while(i<=10);
getch();
return 0;
}
```

Output:

```
1
2
3
4
5
```

Fundamentals of C Programming

- ***Continue statement:***
- Continue statement is a jump statement. The continue statement can be used only inside for loop, while loop and do-while loop.
- Execution of these statement does not cause an exit from the loop but it suspend the execution of the loop for that iteration and transfer control back to the loop for the next iteration.

Fundamentals of C Programming

➤ ***Example for Continue statement:***

```
#include<stdio.h>
#include<conio.h>
int main()
{
int   i ;
clrscr();
for(i=1;i<=5;i++)
{
if(i==3)
continue;
printf("%d\n", i);
}
getch();
return 0;
}
```

Output:

```
1
2

4
5
```

Fundamentals of C Programming

- **Nested Loop:**
- In many cases we may use loop statement inside another looping statement. This type of looping is called nested loop. In nested loop the inner loop is executed first and then outer. The nested loop must be used to input or output multi-dimensional array elements.

Fundamentals of C Programming

- *Example for Nested Loop statement:*

```
#include<stdio.h>
#include<conio.h>
int main()
{
int  i,j;
clrscr();
for(i=1;i<=3,i++)
for(j=1;j<=3;j++)
printf(i=%d\t j=%d\n ", i,j);
getch();
return 0;
}
```

Output:

```
i=1  j=1
i=1  j=2
i=1  j=3
i=2   j=1
i=2  j=2
i=2  j=3
i=3  j=1
i=3  j=2
i=3  j=3
```

Fundamentals of C Programming

- ***Looping Statement :***

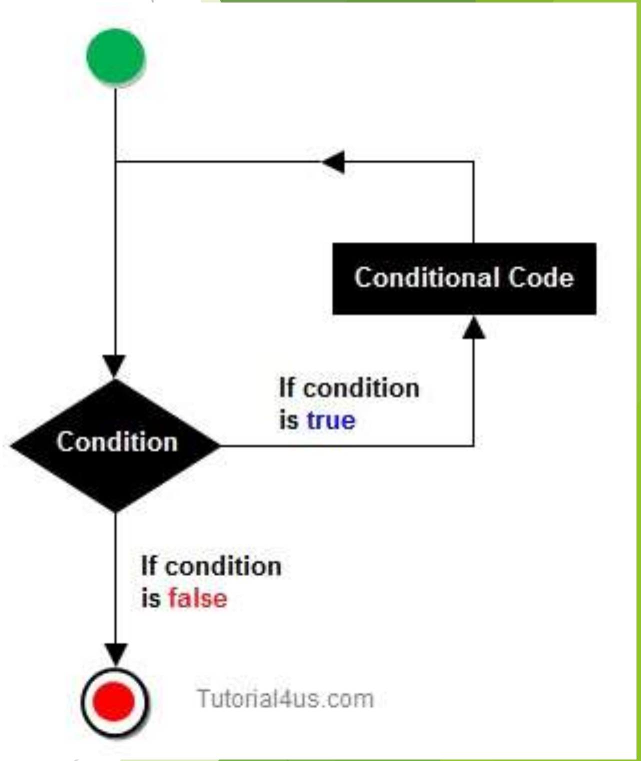
  Looping statement are the statements execute one or more statement repeatedly several number of times. In C programming language there are three types of loops; while, for and do-while.

Why use loop?

When you need to execute a block of code several number of times then you need to use looping concept in C language.

Advantage with looping statement

- Reduce length of Code.
- Take less memory space.
- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.



Fundamentals of C Programming

- ***Types of Loops***

There are three type of Loops available in 'C' programming language:

- while loop
- for loop
- do while

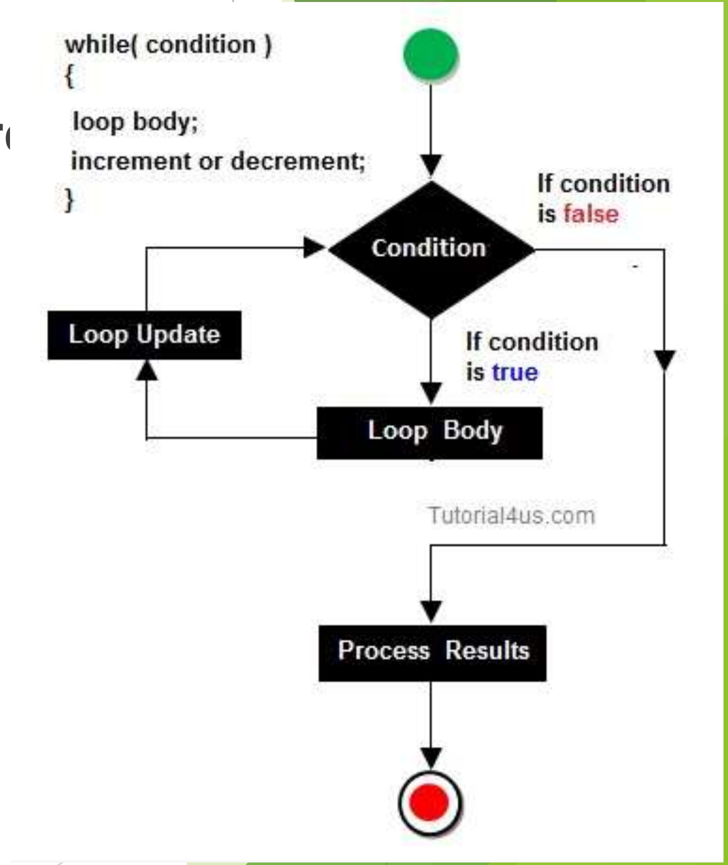
Fundamentals of C Programming

- ***Difference between conditional and looping statement***

  Conditional statement executes only once in the program wher as looping statements executes repeatedly several number of time.

- ***While loop***

  In while loop First check the condition if condition is true then control goes inside the loop body other wise goes outside the body. while loop will be repeats in clock wise direction.



```
while( condition )
{
 loop body;
 increment or decrement;
}
```

Fundamentals of C Programming

- ***Syntax***

Assignment;

while(condition)

{

Statements;

……

Increment/decrements (++ or --);

}

Note: If while loop condition never false then loop become infinite loop

Fundamentals of C Programming

- *Example for while Loop statement:*

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

void main()
{
int i;
clrscr();
i=1;
while(i<5)
{
printf("\n%d",i);
i++;
}
getch();
}
```

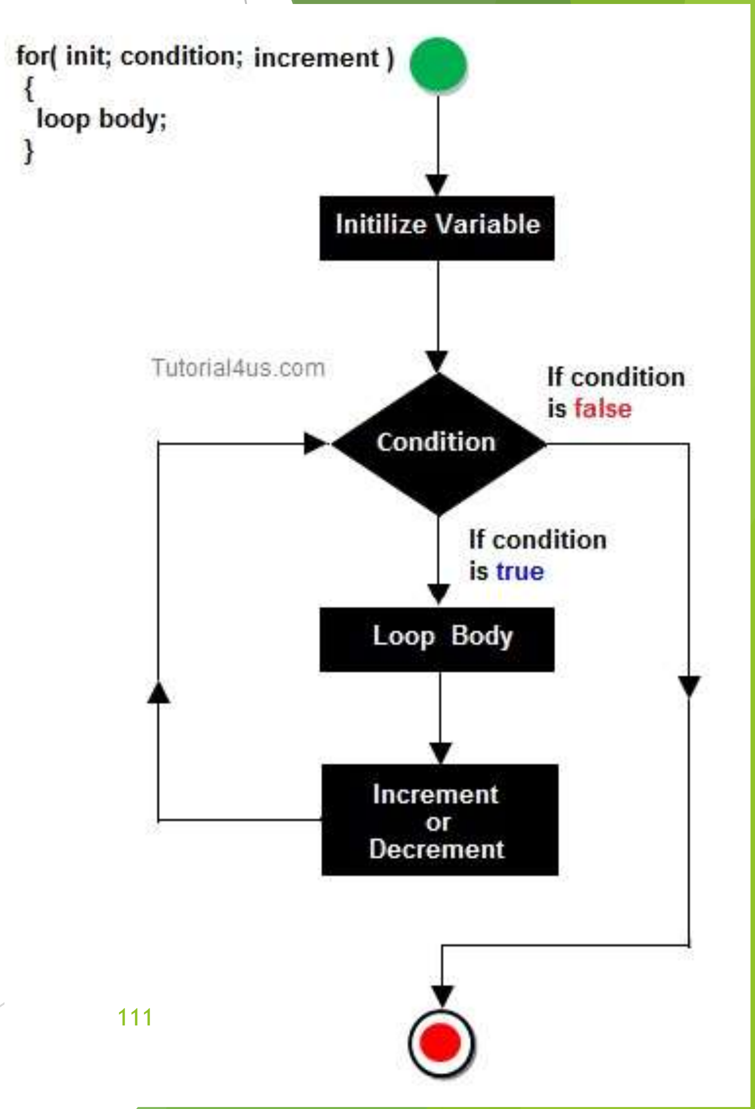
Output:

```
1
2
3
4
```

Fundamentals of C Programming

- **For Loop:**

  for loop is a statement which allows code to be repeatedly executed. For loop contains 3 parts Initialization, Condition and Increment or Decrements.


```
for( init; condition; increment )
{
  loop body;
}
```

Fundamentals of C Programming

- *Example for Loop:*

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

void main()
{
int i;
clrscr();
for(i=1;i<5;i++)
{
printf("\n%d",i);
}
getch();
}
```

Output:

```
1
2
3
4
```

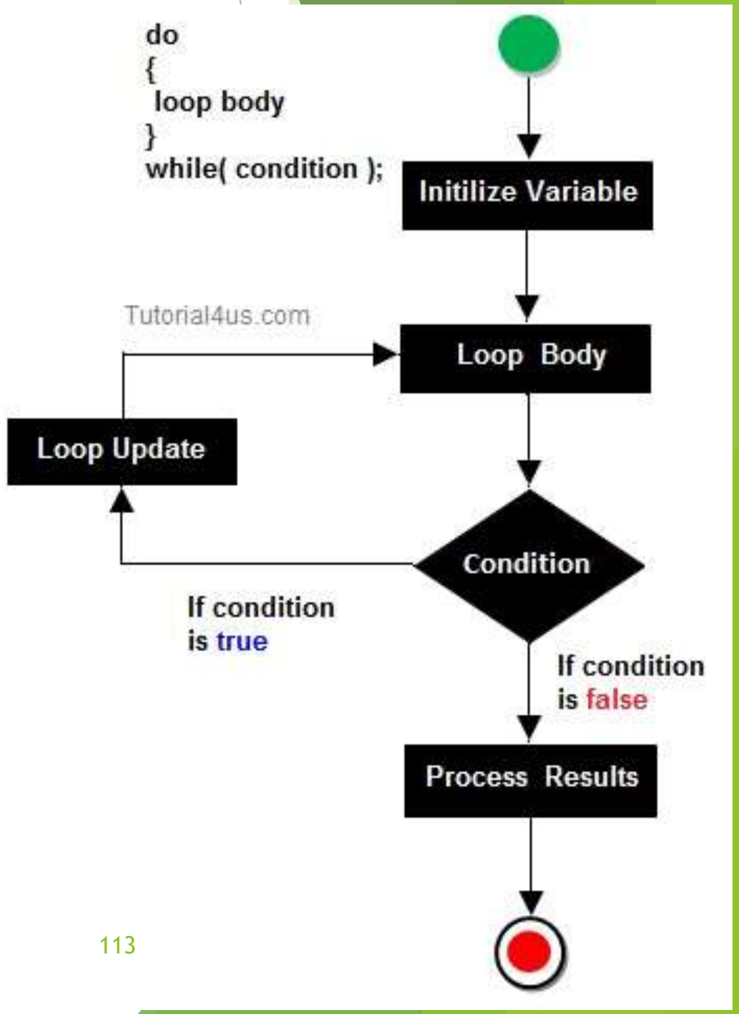
Fundamentals of C Programming

- ***do-while:***

  A do-while loop is similar to a while loop, except that a do-while loop is execute at least one time.
  A do while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).

***Syntax*:**

```
do
{
 Statements;
........
Increment/decrement (++ or --)
} while();
```



Fundamentals of C Programming

- ***Example for When use do..while Loop:***

When we need to repeat the statement block at least 1 time then we use do-while loo

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
clrscr();
i=1;
do
{
printf("\n%d",i);
i++;
}
while(i<5);
getch();
}
```

Output:

```
1
2
3
4
```

Fundamentals of C Programming

- ***Nested Loop*:**

  In Nested loop one loop is place within another loop body.

  When we need to repeated loop body itself n number of times use nested loops.

  Nested loops can be design up to 255 blocks.

Fundamentals of C Programming

Self Assessment Question

1. Who was the father of C language?

A) Bjarne Stroustrup
B) James A. Gosling
C) Dennis Ritchie
D) Dr. E.F. Codd

Answer: **Dennis Ritchie**

Fundamentals of C Programming

Self Assessment Question

2. C Language was developed at _______________.

A) AT & T's Bell Laboratories of USA in 1972
B) AT & T's Bell Laboratories of USA in 1970
C) Sun Microsystems in 1973
D) Cambridge University in 1972

Answer: **AT & T's Bell Laboratories of USA in 1972**

Fundamentals of C Programming

Self Assessment Question

3. For 16bit compiler allowable range for integer constants is ________________.

A) -3.4e38 to 3.4e38
B) -32767 to 32768
C) -32668 to 32667
D) -32768 to 32767

Answer: **-32768 to 32767**

Fundamentals of C Programming

Self Assessment Question

4. C programs are converted into machine language with the help of ____________________.

A) An Editor
B) A Compiler
C) An Operating System
D) None of these

Answer: **A Compiler**

Fundamentals of C Programming

Self Assessment Question

5. C was primarily developed as ____.
    A) System programming language
    B) General purpose language
    C) Data processing language
    D) None of the above

Answer: **System programming language.**

Fundamentals of C Programming

Self Assessment Question

6. Standard ANSI C recognises _____________ number of keywords.

A) 30

B) 32

C) 24

D) 36

E) 40

Answer: **32**

Fundamentals of C Programming

Self Assessment Question

7. Which one of the following is not a reserved keyword for C?

A) auto

B) case

C) main

D) default

E) register

Answer: **main**

Fundamentals of C Programming

Self Assessment Question

8. A C variable cannot start with ____.

A) A number

B) A special symbol other than underscore

C) Both of the above

D) An alphabet

Answer: **Both of the above**

Fundamentals of C Programming

Self Assessment Question

9. Which one of the following is not a valid identifier?

A) _examveda

B) 1examveda

C) exam_veda

D) examveda1

Answer: **1examveda**

Fundamentals of C Programming

Self Assessment Question

10. What is the correct value to return to the operating system upon the successful completion of a program?

A. 1
B. -1
C. 0
D. Program do no return a value
E. 2

Answer: **0**

Fundamentals of C Programming

Self Assessment Question

11. Which is the only function all C programs must contain?

    A. Start()
    B. System ()
    C. Main()
    D. Printf()
    E. getch()

Answer: **main()**

Fundamentals of C Programming

Self Assessment Question

12. Which of the following is not a correct variable type?

    A. Float
    B. real
    C. Int
    D. Double
    E. char

Answer: **real**

Fundamentals of C Programming

Self Assessment Question

13. What number would be shown on the screen after the following statements of C are executed?

```
Char ch;
Int  I;
Ch  =  ‘ G’ ;
I = ch - ‘A’ ;
Printf(“%d” , I );
```

A. 5
B. 6
C. 7
D. 8
E. 9

Answer: **6**

Fundamentals of C Programming

Self Assessment Question

14. Find the output of the following program. Void main() { int i =01289; printf(" %d",i);}

   A. 0289
   B. 1289
   C. 713
   D. 0713
   E. Syntax error

Answer: **Syntax error**

Fundamentals of C Programming

Self Assessment Question

15. Find the output of the following program

```
Void main ()
{
Int i=065, j = 65;
Printf(“%d %d” , i,j);
}
```

a) 53 65

b) 65 65

c) 065 65

d) 053 65

e) Syntax error

Answer: **53 65**

Fundamentals of C Programming

Document Links

Topics	URL	Notes
Overview of C	https://www.tutorialspoint.com/cprogramming/c_overview.htm **https://www.sitesbay.com/cprogramming/c-features**	This link explains the definition and overview of C programming
Data Types	http://c-language.com/c-tutorial/c-datatypes/ https://www.studytonight.com/c/datatype-in-c.php	This link explains the datatypes and their types
Constants	https://www.tutorialspoint.com/cprogramming/c_constants.htm https://fresh2refresh.com/c-programming/c-constants/	This link guides about the constants
Variables	http://c-language.com/c-tutorial/c-variables/	This link gives a detail about the variables with supporting examples
Operators	http://c-language.com/c-tutorial/c-operators/	This link describes the various types of operators with syntax guide
Expression	http://c-language.com/c-tutorial/c-expressions/	This link describes about the expression with examples
Type casting	https://www.developerinsider.in/type-casting-c-programming/ https://www.geeksforgeeks.org/type-conversion-c/ https://www.improgrammer.net/type-casting-c-language/	This link explains the completely about the type casting
Controls Statements	https://www.programiz.com/c-programming/c-if-else-statement	This links guides about various types of control statements in details
Looping Statements	https://www.sitesbay.com/cprogramming/c-looping-statement	This link explains about the concept and types

Fundamentals of C Programming

Video Links

Topics	URL	Notes
Overview of C programming	https://www.youtube.com/watch?v=05nhvJnTkEU	The video gives an overview on the concept of C programming
Data Types	https://www.youtube.com/watch?v=j1u3V6pzwEI https://www.youtube.com/watch?v=bS6uNMmIoQ0	The video gives an overview on the data types in c programming
Looping	https://www.youtube.com/watch?v=_NZZ9Y-j_mw	This video explains the about the looping and control statements
Operators and Expression	https://www.youtube.com/watch?v=ajH54_s7nuc	This video explains the about the operators and expression
Statements	https://www.youtube.com/watch?v=0lKWFArcTGU	This video explains the about the all the statements

Fundamentals of C Programming

EBook's Links

Topics	URL	Page Number
Overview of C	https://www.cluster2.hostgator.co.in/files/writeable/uploads/hostgator99706/file/letusc-yashwantkanetkar.pdf	Page Number 2 - Page Number 13
Data Types	http://www.skiesanduniverses.org/resources/The_C_Programming_Language.pdf	Page Number 25 - Page Number 53
Type casting	http://www-personal.acfr.usyd.edu.au/tbailey/ctext/ctext.pdf	Page Number 8 - Page Number 10
Operators	http://www.dipmat.univpm.it/~demeio/public/the_c_programming_language_2.pdf	Page Number 35 – Page Number 46
Expression	http://www.dipmat.univpm.it/~demeio/public/the_c_programming_language_2.pdf	Page Number 50 - Page Number 52
Controls Statements / Looping Statements	http://www.dipmat.univpm.it/~demeio/public/the_c_programming_language_2.pdf	Page Number 55 - Page Number 65

Fundamentals of C Programming

Assignment

Write 10 different types of C programs.

Topics :

- Odd and even
- Integer numbers
- Find the greater than 5
- Find the positive numbers
- Find the alphabets
- Find the missing letters