DIRECTORATE OF DISTANCE EDUCATION

UNIVERSITY OF NORTH BENGAL

MASTER OF SCIENCES- MATHEMATICS

SEMESTER -IV

NUMERICAL PROBLEM SOLVING BY COMPUTER PROGRAMMING (THEORY)

DEMATH4SCORE2

BLOCK-1

UNIVERSITY OF NORTH BENGAL

PostalAddress:

The Registrar,

University of North Bengal,

Raja Rammohunpur,

P.O.-N.B.U.,Dist-Darjeeling,

West Bengal, Pin-734013,

India.

Phone: (O) +91 0353-2776331/2699008

Fax:(0353) 2776313, 2699001

Email: regnbu@sancharnet.in ; regnbu@nbu.ac.in

Wesbsite: www.nbu.ac.in

First Published in 2019

FOREWORD

The Self Learning Material (SLM) is written with the aim of providing simple and organized study content to all the learners. The SLMs are prepared on the framework of being mutually cohesive, internally consistent and structured as per the university’s syllabi. It is a humble attempt to give glimpses of the various approaches and dimensions to the topic of study and to kindle the learner’s interest to the subject

We have tried to put together information from various sources into this book that has been written in an engaging style with interesting and relevant examples. It introduces you to the insights of subject concepts and theories and presents them in a way that is easy to understand and comprehend.

We always believe in continuous improvement and would periodically update the content in the very interest of the learners. It may be added that despite enormous efforts and coordination, there is every possibility for some omission or inadequacy in few areas or topics, which would definitely be rectified in future.

We hope you enjoy learning from this book and the experience truly enrich your learning and help you to advance in your career and future endeavours.

NUMERICAL PROBLEM SOLVING BY COMPUTER PROGRAMMING

BLOCK-1 NUMERICAL PROBLEM SOLVING BY COMPUTER PROGRAMMING

Introduction to Block

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc ANSI C standard emerged in the early 1980s, this book was split into two titles: The original was still called Programming in C, and the title that covered ANSI C was called Programming in ANSI C. This was done because it took several years for the compiler vendors to release their ANSI C compilers and for them to become ubiquitous.

Understand the different elements of C language like C Variable, C Constant, Character Set, Tokens, Keywords and Identifiers .C programming language provides built-in functions to perform input operations to read any given input and to display data on screen when there is a need to output the result. Understand the Expression and Expression Types in C, Understand the concept of C Operator Precedence and Associativity. Branching is deciding what actions to take and looping is deciding how many times to take a certain action. Structured programming is a programming technique in which a larger program is divided into smaller subprograms to make it easy to understand, easy to implement and makes the code reusable, etc. Understand the Functions and types of Functions in C

UNIT-1 COMPUTER PROGRAMMING

STRUCTURE

1.0 OBJECTIVES

Understand the basics of Computer programming

Understand types of Computing Environment

Comprehend the Modular programming

Enumerate the Program development cycle

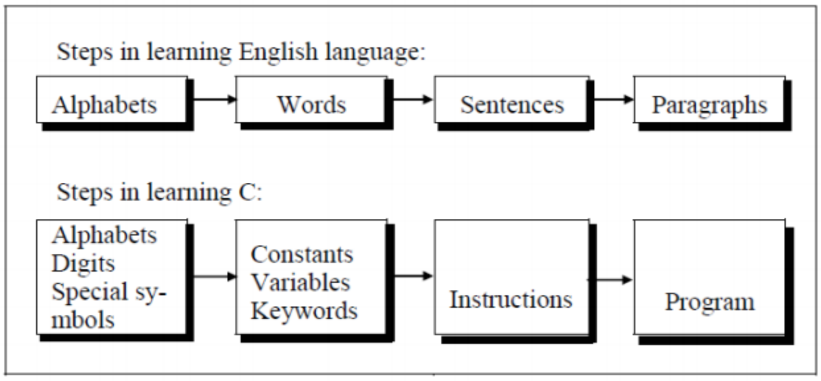
1.1 INTRODUCTION

Computer programming is the act of writing computer programs, which are a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer.
Computer Programming is fun and easy to learn provided you adopt a proper approach. This tutorial attempts to cover the basics of computer programming using a simple and practical approach for the benefit of novice learners.

1.2 OVERVIEW OF COMPUTER PROGRAMMING LANGUAGES

There is a close analogy between learning English language and learning C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to form sentences and sentences are combined to form paragraphs. Learning C is similar and easier. Instead of straight-away learning how to write programs, we must first know what alphabets, numbers and special symbols are used in C, then how using them constants, variables and keywords are constructed, and finally how are these combined to form an instruction. A group of instructions would be combined later on to form a program. So a computer program is just a collection of the instructions necessary to solve a specific problem. The basic operations of a computer system form what is known as the computer’s instruction set. And the approach or method that is used to solve the problem is known as an algorithm.
Such instructions can be executed directly when they are in the computer manufacturer-specific numerical form known as machine language, after a simple substitution process when expressed in a corresponding assembly language, or after translation from some “higher-level” language. Although there are many computer languages, relatively few are widely used.

Machine and assembly languages are “low-level,” requiring a programmer to manage explicitly all of a
computer’s idiosyncratic features of data storage and operation. In contrast, high-level languages shield a programmer from worrying about such considerations and provide a notation that is more easily written and read by programmers.


Steps in learning English language:
Alphabets
Words
Sentences
Paragraphs
Steps in learning C:
Alphabets Digits Special sy-mbols
Constants Variables Keywords
Instructions
Program

So for as programming language concern these are of two types.

1) Low level language

2) High level language

1.3 COMPUTING ENVIRONMENTS

What is Computing Environment?

When we want to solve a problem using a computer, the computer makes use of various devices which work together to solve that problem. There may be a various number of ways to solve a problem. We use the various number of computer devices arranged in different ways to solve different problems. The arrangement of computer devices to solve a problem is said to be a computing environment. The formal definition of the computing environment is as follows.

Computing Environment is a collection of computers which are used to process and exchange information to solve various types of computing problems.

1.3.1 Types of Computing Environments

The following are the various types of computing environments.

1. Personal Computing Environment

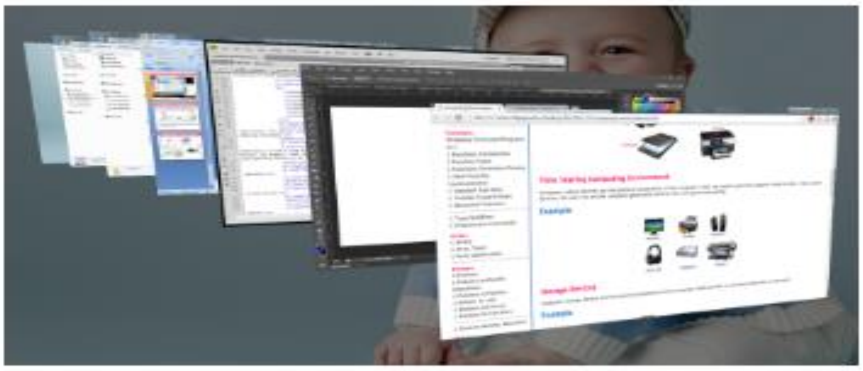
2. Time-Sharing Computing Environment
3. Client-Server Computing Environment
4. Distributed Computing Environment
5. Grid Computing Environment
6. Cluster Computing Environment

Personal Computing Environment

Personal computing is a stand-alone machine. In a personal computing environment, the complete program resides on the stand-alone machine and executed from the same machine. Laptops, mobile devices, printers, scanners and the computer systems we use at home, office are the examples for the personal computing environment.

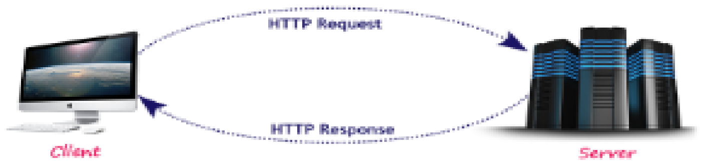
Time-Sharing Computing Environment

The time-sharing computing environment is a stand-alone computer in which a single user can perform multiple operations at a time by using a multitasking operating system. Here the processor time is divided among different tasks and this is called “Time-sharing”. For example, a user can listen to music while writing something in a text editor. Windows 95 and later versions of Windows OS, iOS and Linux operating systems are the examples for this computing environment.



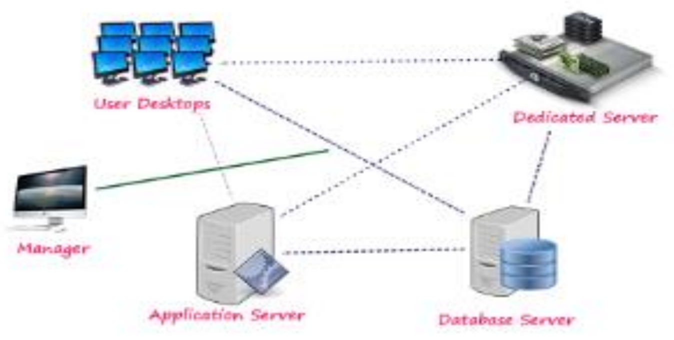
Client-Server Computing Environment

The client-server environment contains two machines (Client machine and Server machine). These both machines will exchange the information through an application. Here Client is a normal computer like PC, Tablet, Mobile, etc., and Server is a powerful computer which stores huge data and manages the huge amount of file and emails, etc., In this environment, client requests for data and server provides data to the client. In the client-server environment, the communication between client and server is performed using HTTP (Hyper Text Transfer Protocol).



Distributed Computing Environment

In the distributed computing environment, the complete functionality of the software is not on a single computer but is distributed among multiple computers. Here we use a method of computer processing in which different programs of an application run simultaneously on two or more computers. These computers communicate with each other over a network to perform the complete task. In a distributed computing environment, the data is distributed among different systems and that data is logically related to each other.



Grid Computing Environment

Grid computing is a collection of computers from different locations. All these computers work for a common problem. A grid can be described as a distributed collection of a large number of computers working for a single application.

Cluster Computing Environment

Cluster computing is a collection of interconnected computers. These computers work together to solve a single problem. In a cluster computing environment, a collection of systems work together as a single system.

1.4 TYPES OF COMPUTER PROGRAM LANGUAGE

Machine and assembly languages

A machine language consists of the numeric codes for the operations that a particular computer can execute directly. The codes are strings of 0s and 1s, or binary digits (“bits”), which are frequently converted both from and to hexadecimal (base 16) for human viewing and modification. Machine language instructions typically use some bits to represent operations, such as addition, and some to represent operands, or perhaps the location of the next instruction. Machine language is difficult to read and write, since it does not resemble conventional mathematical notation or human language, and its codes vary from computer to computer.

Assembly language is one level above machine language. It uses short mnemonic codes for instructions and allows the programmer to introduce names for blocks of memory that hold data. One might thus write "add pay, total" instead of "0110101100101000" for an instruction that adds two numbers.

Assembly language is designed to be easily translated into machine language. Although blocks of data may be referred to by name instead of by their machine addresses, assembly language does not provide more sophisticated means of organizing complex information. Like machine language, assembly language requires detailed knowledge of internal computer architecture. It is useful when such details are important, as in programming a computer to interact with input/output devices (printers, scanners, storage devices, and so forth)

Algorithmic languages

Algorithmic languages are designed to express mathematical or symbolic computations. They can express algebraic operations in notation similar to mathematics and allow the use of subprograms that package commonly used operations for reuse. They were the first high-level languages

Business-oriented languages

COBOL (*co*mmon *b*usiness *o*riented *l*anguage) has been heavily used by businesses since its inception in 1959. A committee of computer manufacturers and users and U.S. government organizations established CODASYL (*Co*mmittee on *Da*ta *Sy*stems and *L*anguages) to develop and oversee the language standard in order to ensure its portability across diverse systems.

COBOL uses an English-like notation—novel when introduced. Business computations organize and manipulate large quantities of data, and COBOL introduced the record data structure for such tasks. A record clusters heterogeneous data such as a name, ID number, age, and address into a single unit. This contrasts with scientific languages, in which homogeneous arrays of numbers are common. Records are an important example of "chunking" data into a single object, and they appear in nearly all modern languages.

Visual Basic

Visual Basic was developed by Microsoft to extend the capabilities of BASIC by adding objects and “event-driven” programming: buttons, menus, and other elements of graphical user interfaces (GUIs). Visual Basic can also be used within other Microsoft software to program small routines.

Declarative languages

Declarative languages, also called nonprocedural or very high level, are programming languages in which (ideally) a program specifies what is to be done rather than how to do it. In such languages there is less difference between the specification of a program and its implementation than in the procedural languages described so far. The two common kinds of declarative languages are logic and functional languages.

Check your Progress-1

1. Define Computing Environment. State its Types

____

____

____

2. What is Declarative language?

____

____

____

1.5 MODULAR PROGRAMMING

Modular programming is the process of subdividing a computer program into separate sub-programs.

A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system. Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.

Object-oriented programming (OOP) is compatible with the modular programming concept to a large extent. Modular programming enables multiple programmers to divide up the work and debug pieces of the program independently.

Modules in modular programming enforce logical boundaries between components and improve maintainability. They are incorporated through interfaces. They are designed in such a way as to minimize dependencies between different modules. Teams can develop modules separately and do not require knowledge of all modules in the system.

Each and every modular application has a version number associated with it. This provides developers flexibility in module maintenance. If any changes have to be applied to a module, only the affected subroutines have to be changed. This makes the program easier to read and understand.

Modular programming has a main module and many auxiliary modules. The main module is compiled as an executable (EXE), which calls the auxiliary module functions. Auxiliary modules exist as separate executable files, which load when the main EXE runs. Each module has a unique name assigned in the PROGRAM statement. Function names across modules should be unique for easy access if functions used by the main module must be exported.

Languages that support the module concept are IBM Assembler, COBOL, RPG, FORTRAN, Morpho, Zonnon and Erlang, among others.

The benefits of using modular programming include:

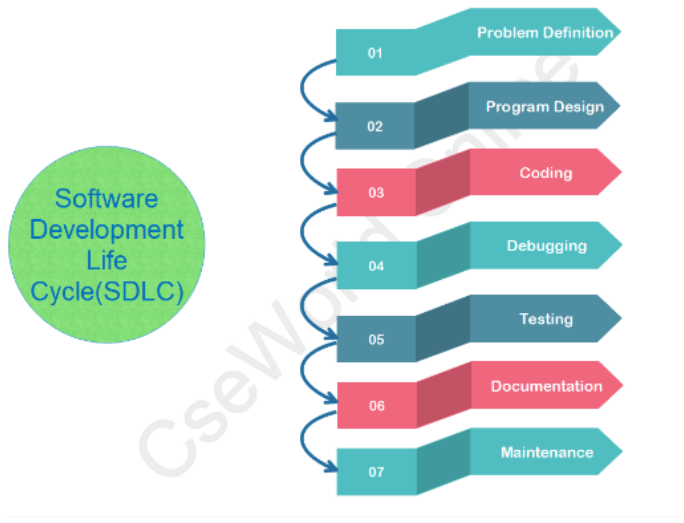
- Less code has to be written.
- A single procedure can be developed for reuse, eliminating the need to retype the code many times.
- Programs can be designed more easily because a small team deals with only a small part of the entire code.
- Modular programming allows many programmers to collaborate on the same application.
- The code is stored across multiple files.

- Code is short, simple and easy to understand.
- Errors can easily be identified, as they are localized to a subroutine or function.
- The same code can be used in many applications.
- The scoping of variables can easily be controlled.

1.6 PROGRAM DEVELOPMENT CYCLE

The various stages in the development of a computer program are :

1. Problem Definition
2. Program Design
3. Coding
4. Debugging
5. Testing
6. Documentation
7. Maintenance



Problem Definition:

- The first step in the process of program development is the thorough understanding and identification of the problem for which is the program or software is to be developed.
- In this step the problem has to be defined formally.

- All the factors like Input/output, processing requirement, memory requirements, error handling, interfacing with other programs have to be taken into consideration in this stage.

Program Design:

- The next stage is the program design. The software developer makes use of tools like algorithms and flowcharts to develop the design of the program.
  - Algorithm
  - Flowchart

Coding:

- Once the design process is complete, the actual computer program is written, i.e. the instructions are written in a computer language.
- Coding is generally a very small part of the entire program development process and also a less time consuming activity in reality.
- In this process all the syntax errors i.e. errors related to spelling, missing commas, undefined labels etc. are eliminated.
- For effective coding some of the guide lines which are applied are :
  - Use of meaningful names and labels of variables,
  - Simple and clear expressions,
  - Modularity with emphasis on making modules generalized,
  - Making use of comments and indenting the code properly,
  - Avoiding jumps in the program to transfer control.

Debugging:

- At this stage the errors in the programs are detected and corrected.
- This stage of program development is an important process. Debugging is also known as program validation.
- Some common errors which might occur in the programs include:
  - Un initialization of variables.
  - Reversing of order of operands.
  - Confusion of numbers and characters.

    - Inverting of conditions eg jumping on zero instead of on not zero.

Testing:

- The program is tested on a number of suitable test cases.
- A test plan of the program has to be done at the stage of the program design itself.
- This ensures a thorough understanding of the specifications.
- The most trivial and the most special cases should be identified and tested.
- It is always useful to include the maximum and minimum values of all variables as test data.

Documentation:

- **Documentation is a very essential step in the program** development.
- Documentation help the users and the people who maintain the software.
- This ensures that future modification if required can be done easily. Also it is required during redesigning and maintenance.

Maintenance:

- Updating and correction of the program for changed conditions and field experience is accounted for in maintenance.
- Maintenance becomes essential in following situations:
    - Change in specification,
    - Change in equipment,
    - Errors which are found during the actual execution of the program.

Computer Programming Languages are also made of several elements. These basic elements include −

- Programming Environment
- Basic Syntax
- Data Types
- Variables
- Keywords
- Basic Operators

- Decision Making
- Loops
- Numbers
- Characters
- Arrays
- Strings
- Functions
- File I/O

1.7 WHAT IS C PROGRAMMING

C is a general-purpose programming language that is extremely popular, simple and flexible. It is machine-independent, structured programming language which is used extensively in various applications.

C was the basics language to write everything from operating systems (Windows and many others) to complex programs like the Oracle database, Git, Python interpreter and more.

It is said that 'C' is a god's programming language. One can say, C is a base for the programming. If you know 'C,' you can easily grasp the knowledge of the other programming languages that uses the concept of 'C'

It is essential to have a background in computer memory mechanisms because it is an important aspect when dealing with the C programming language.

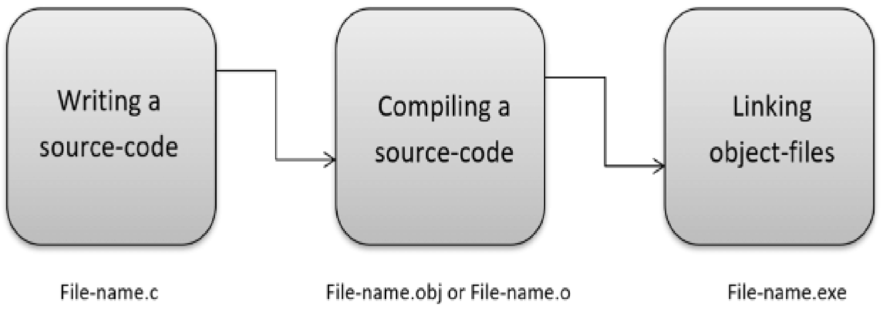
1.7.1 Where is C used? Key Applications

1. 'C' language is widely used in embedded systems.
2. It is used for developing system applications.
3. It is widely used for developing desktop applications.
4. Most of the applications by Adobe are developed using 'C' programming language.
5. It is used for developing browsers and their extensions. Google's Chromium is built using 'C' programming language.
6. It is used to develop databases. MySQL is the most popular database software which is built using 'C'.

7. It is used in developing an operating system. Operating systems such as Apple's OS X, Microsoft's Windows, and Symbian are developed using 'C' language. It is used for developing desktop as well as mobile phone's operating system.
8. It is used for compiler production.
9. It is widely used in IOT applications.

1.7.2 How 'C' Works?

C is a compiled language. A compiler is a special tool that compiles the program and converts it into the object file which is machine readable. After the compilation process, the linker will combine different object files and creates a single executable file to run the program. The following diagram shows the execution of a 'C' program



Nowadays, various compilers are available online, and you can use any of those compilers. The functionality will never differ and most of the compilers will provide the features required to execute both 'C' and 'C++' programs.

Following is the list of popular compilers available online:

- Clang compiler
- MinGW compiler (Minimalist GNU for Windows)
- Portable 'C' compiler
- Turbo C

Check your Progress-2

3. Explain Debugging in Program Development Cycle.

4. State key application of C.

1.7 LET US SUM UP

Computer languages allow computers to quickly and efficiently process large and complex swaths of information. Computer programming helps in developing programming languages which are used for transforming computing problems into instructions. Programming languages have helped in the development of the internet and mobile applications that has changed life of humans greatly.

1.8 KEYWORDS

Operating system: the low-level software that supports a computer's basic functions, such as scheduling tasks and controlling peripherals.

Embedded system: An **embedded system** is a combination of computer hardware and software, either fixed in capability or programmable, designed for a specific function or functions within a larger **system**

Compiler - is a special tool that compiles the program and converts it into the object file which is machine readable.

1.9 QUESTIONS FOR REVIEW

1. What do you understand by Machine Assembly Language?

2. Enumerate Program Development Cycle

3.Explain the working of C.

1.10 SUGGESTED READINGS AND REFERENCES

1. B. Gottfried: Programming with C , Tata McGraw-Hill Edition 2002.
2. E. Balagurusamy : Programming in ANSI C, Tata Mcgraw Hill - Edition 2002.
3. Brain W. Kernighan & Dennis M. Ritchie, The C Programme Language, 2nd Edition (ANSI features) , Prentice Hall 1989.
4. Let Us C- Y.P. Kanetkar, BPB Publication - 2002.
5. Analysis of Numerical Methods—Isacsons& Keller.
6. Numerical solutions of Ord. Diff. Equations—M K Jain
7. Numerical solutions of Partial Diff. Equations—G D Smith.
8. Programming with C, B. Gottfried, Tata-McGraw Hill
9. Programming with C, K. R. Venugopal and Sudeep R. Prasad, Tata-McGraw Hill

1.11 ANSWERS TO CHECK YOUR PROGRESS

1. Provide definition- 1.3 and types – 1.3.1

2. Provide explanation --1.4

3. Provide explanation of debugging – 1.6

4. Provide applications – 1.7.1

UNIT-2 INTRODUCTION TO C LANGUAGE

STRUCTURE

2.0 OBJECTIVES

Understand the **C** Program Basics

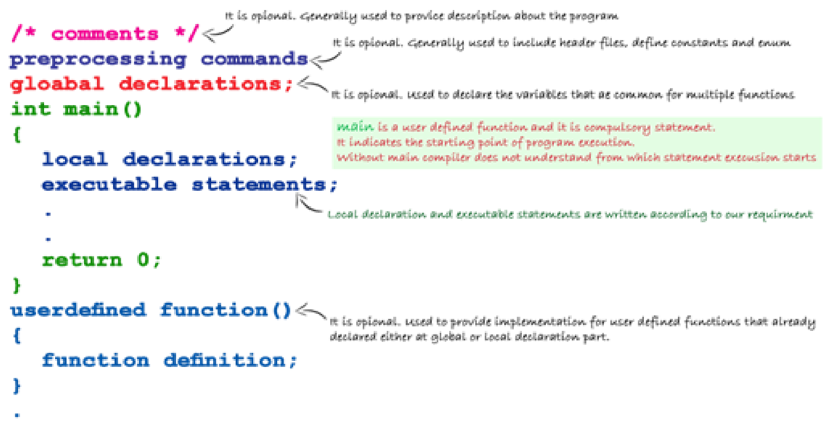
Understand the elements of C - Character Set, Tokens, Keywords and Identifiers Comprehend **C** data types

2.1 INTRODUCTION

C is a structured programming language. It is also known as function orientated programming language. C programming language was developed in the year of **1972** by **Dennis Ritchie** at Bell Laboratories in the USA (AT & T).
In the year of 1968, research was started by Dennis Ritchie on programming languages like BCPL, CPL. The main aim of his research was to develop a new language to create an OS called UNIX. After four years of research, a new programming language was created with solutions for drawbacks in languages like BCPL & CPL. In the year of 1972, the new language was introduced with the name "**Traditional C**".

2.2 C PROGRAM BASICS

C is a structured programming language. Every C program and its statements must be in a particular structure. Every C program has the following general structure...



```
/* comments */
preprocessing commands
gloabal declarations;
int main()
{
   local declarations;
   executable statements;
   .
   .
   return 0;
}
userdefined function()
{
   function definition;
}
.
```

It is oplonal. Generally used to provice description about the program

It is oplonal. Generally used to include header files, define constants and enum

It is opional. used to declare the variables that ae common for multiple functions

main is a user defined function and it is compulsory statement.
It indicates the starting point of program execution.
Without main compiler does not understand from which statement execusion starts

Local declaration and executable statements are written according to our requirment

It is opional. used to provide implementation for user defined functions that already declared either at global or local declaration part.

Line 1: Comments - They are ignored by the compiler

This section is used to provide a small description of the program. The comment lines are simply ignored by the compiler, that means they are not executed. In C, there are two types of comments.

1. **Single Line Comments:** Single line comment begins with // symbol. We can write any number of single line comments.
2. **Multiple Lines Comments:** Multiple lines comment begins with /* symbol and ends with */. We can write any number of multiple lines comments in a program.

In a C program, the comment lines are optional. Based on the requirement, we write comments. All the comment lines in a C program just provide the guidelines to understand the program and its code.

Line 2: Preprocessing Commands

Preprocessing commands are used to include header files and to define constants. We use the **#include** statement to include the header file into our program. We use a **#define** statement to define a constant. The preprocessing statements are used according to the requirements. If we don't need any header file, then no need to write #include statement. If we don't need any constant, then no need to write a #define statement.

Line 3: Global Declaration

The global declaration is used to define the global variables, which are common for all the functions after its declaration. We also use the global declaration to declare functions. This global declaration is used based on the requirement.

Line 4: int main()

Every C program must write this statement. This statement (main) specifies the starting point of the C program execution. Here, main is a user-defined method which tells the compiler that this is the starting point of the program execution. Here, **int** is a data type of a value that is going to return to the Operating System after completing the main method execution. If we don't want to return any value, we can use it as **void**.

Line 5: Open Brace ({)

The open brace indicates the beginning of the block which belongs to the main method. In C program, every block begins with a '{' symbol.

Line 6: Local Declaration

In this section, we declare the variables and functions that are local to the function or block in which they are declared. The variables which are declared in this section are valid only within the function or block in which they are declared.

Line 7: Executable statements

In this section, we write the statements which perform tasks like reading data, displaying the result, calculations, etc., All the statements in this section are written according to the requirements.

Line 9: Closing Brace (})

The close brace indicates the end of the block which belongs to the main method. In C program every block ends with a '}' symbol.

Line 10, 11, 12, ...: User-defined function()

This is the place where we implement the user-defined functions. The user-defined function implementation can also be performed before the main method. In this case, the user-defined function need not be declared. Directly it can be implemented, but it must be before the main method. In a program, we can define as many user-defined functions as we want. Every user-defined function needs a function call to execute its statements.

2.2.1 General rules for any C program

1. Every executable statement must end with a semicolon symbol (;).
2. Every C program must contain exactly one main method (Starting point of the program execution).

3. All the system-defined words (keywords) must be used in lowercase letters.
4. Keywords can not be used as user-defined names(identifiers).
5. For every open brace ({), there must be respective closing brace (}).
6. Every variable must be declared before it is used.

2.3 C CHARACTER SET

As every language contains a set of characters used to construct words, statements, etc., C language also has a set of characters which include **alphabets, digits**, and **special symbols**. C language supports a total of 256 characters.

Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C language character set contains the following set of characters...

1. Alphabets
2. Digits
3. Special Symbols

Alphabets

C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.

lower case letters - **a to z**

UPPER CASE LETTERS - **A to Z**

Digits

C language supports 10 digits which are used to construct numerical values in C language.

Digits - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

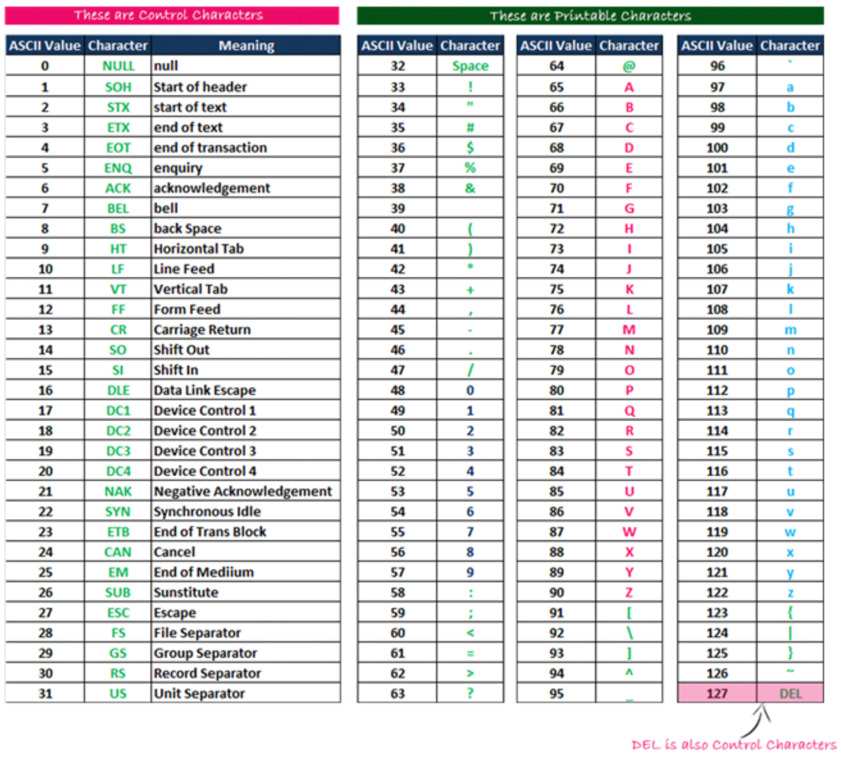
Special Symbols

C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.

Special Symbols - **~ @ # $ % ^ & * ( ) _ - + = { } [ ] ; : ' " / ? . > , < \ |**

tab newline space NULL bell backspace verticaltab etc.,

Every character in C language has its equivalent ASCII (American Standard Code for Information Interchange value



These are Control Characters			These are Printable Characters					
ASCII Value	Character	Meaning	ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
0	NULL	null	32	Space	64	@	96	`
1	SOH	Start of header	33	!	65	A	97	a
2	STX	start of text	34	"	66	B	98	b
3	ETX	end of text	35	#	67	C	99	c
4	EOT	end of transaction	36	$	68	D	100	d
5	ENQ	enquiry	37	%	69	E	101	e
6	ACK	acknowledgement	38	&	70	F	102	f
7	BEL	bell	39		71	G	103	g
8	BS	back Space	40	(	72	H	104	h
9	HT	Horizontal Tab	41	)	73	I	105	i
10	LF	Line Feed	42	*	74	J	106	j
11	VT	Vertical Tab	43	+	75	K	107	k
12	FF	Form Feed	44	,	76	L	108	l
13	CR	Carriage Return	45	-	77	M	109	m
14	SO	Shift Out	46	.	78	N	110	n
15	SI	Shift In	47	/	79	O	111	o
16	DLE	Data Link Escape	48	0	80	P	112	p
17	DC1	Device Control 1	49	1	81	Q	113	q
18	DC2	Device Control 2	50	2	82	R	114	r
19	DC3	Device Control 3	51	3	83	S	115	s
20	DC4	Device Control 4	52	4	84	T	116	t
21	NAK	Negative Acknowledgement	53	5	85	U	117	u
22	SYN	Synchronous Idle	54	6	86	V	118	v
23	ETB	End of Trans Block	55	7	87	W	119	w
24	CAN	Cancel	56	8	88	X	120	x
25	EM	End of Mediium	57	9	89	Y	121	y
26	SUB	Sunstitute	58	:	90	Z	122	z
27	ESC	Escape	59	;	91	[	123	{
28	FS	File Separator	60	<	92	\	124	|
29	GS	Group Separator	61	=	93	]	125	}
30	RS	Record Separator	62	>	94	^	126	~
31	US	Unit Separator	63	?	95	_	127	DEL

DEL is also Control Characters

Check your Progress-1

1. Define Multiple Lines Comments.

____

____

____

2. Explain special symbols used in C

____

____

____

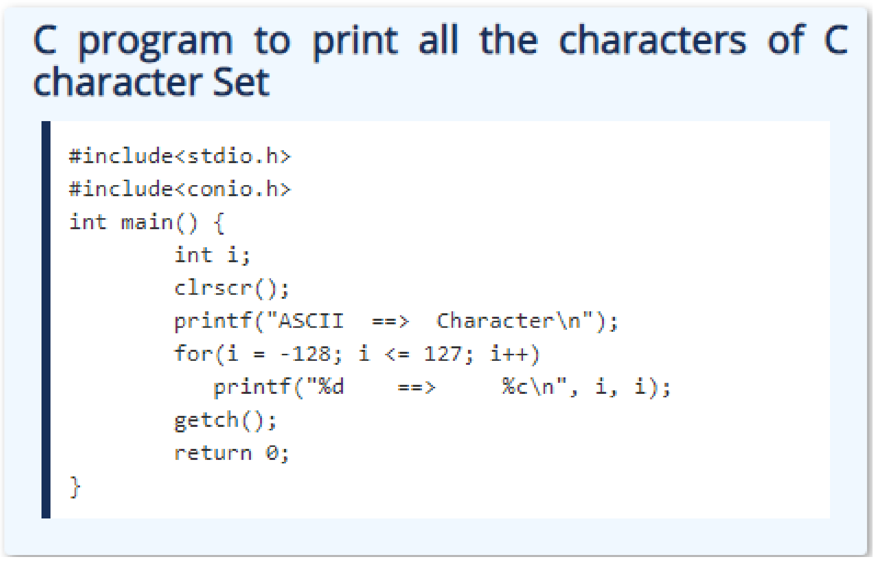
2.4 C TOKENS

Every C program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a c program is called token. Every instruction in a C program is a collection of tokens. Tokens are used to construct C programs and they are said to the basic building blocks of a C program.
In a C program tokens may contain the following...

1. Keywords
2. Identifiers
3. Operators
4. Special Symbols
5. Constants
6. Strings
7. Data values

In a C program, a collection of all the keywords, identifiers, operators, special symbols, constants, strings, and data values are called tokens

Consider the following C program...



C program to print all the characters of C character Set

```
#include<stdio.h>
#include<conio.h>
int main() {
        int i;
        clrscr();
        printf("ASCII  ==>  Character\n");
        for(i = -128; i <= 127; i++)
           printf("%d    ==>    %c\n", i, i);
        getch();
        return 0;
}
```

In the above program there are 22 tokens.

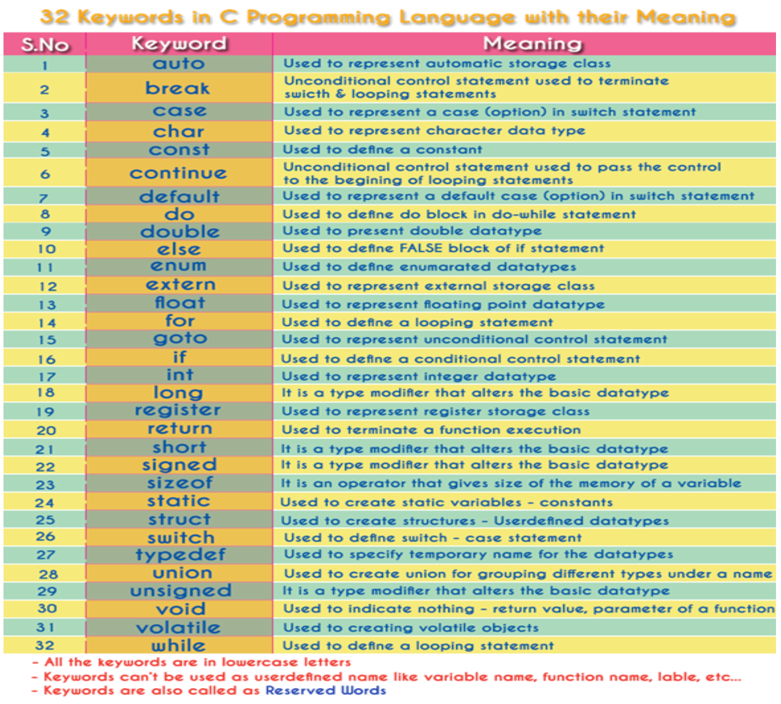
2.5 C KEYWORDS

As every language has words to construct statements, C programming also has words with a specific meaning which are used to construct c program instructions. In the C programming language, keywords are special words with predefined meaning. Keywords are also known as reserved words in C programming language.
In the C programming language, there are **32 keywords**. All the 32 keywords have their meaning which is already known to the compiler
Keywords are the reserved words with predefined meaning which already known to the compiler Whenever C compiler come across a keyword, automatically it understands its meaning.

2.5.1 Properties of Keywords

1. All the keywords in C programming language are defined as lowercase letters so they must be used only in lowercase letters
2. Every keyword has a specific meaning, users can not change that meaning.
3. Keywords cannot be used as user-defined names like variable, functions, arrays, pointers, etc...
4. Every keyword in C programming language represents something or specifies some kind of action to be performed by the compiler.

The following table specifies all the 32 keywords with their meaning...



32 Keywords in C Programming Language with their Meaning

S.No	Keyword	Meaning
1	auto	Used to represent automatic storage class
2	break	Unconditional control statement used to terminate swicth & looping statements
3	case	Used to represent a case (option) in switch statement
4	char	Used to represent character data type
5	const	Used to define a constant
6	continue	Unconditional control statement used to pass the control to the begining of looping statements
7	default	Used to represent a default case (option) in switch statement
8	do	Used to define do block in do-while statement
9	double	Used to present double datatype
10	else	Used to define FALSE block of if statement
11	enum	Used to define enumarated datatypes
12	extern	Used to represent external storage class
13	float	Used to represent floating point datatype
14	for	Used to define a looping statement
15	goto	Used to represent unconditional control statement
16	if	Used to define a conditional control statement
17	int	Used to represent integer datatype
18	long	It is a type modifier that alters the basic datatype
19	register	Used to represent register storage class
20	return	Used to terminate a function execution
21	short	It is a type modifier that alters the basic datatype
22	signed	It is a type modifier that alters the basic datatype
23	sizeof	It is an operator that gives size of the memory of a variable
24	static	Used to create static variables - constants
25	struct	Used to create structures - Userdefined datatypes
26	switch	Used to define switch - case statement
27	typedef	Used to specify temporary name for the datatypes
28	union	Used to create union for grouping different types under a name
29	unsigned	It is a type modifier that alters the basic datatype
30	void	Used to indicate nothing - return value, parameter of a function
31	volatile	Used to creating volatile objects
32	while	Used to define a looping statement

- All the keywords are in lowercase letters
- Keywords can't be used as userdefined name like variable name, function name, lable, etc...
- Keywords are also called as Reserved Words

Check your Progress-2

3. What is Token in C?

___

___

___

4. What do you understand by keywords?

___

___

___

2.6 C IDENTIFIERS

In C programming language, programmers can specify their name to a variable, array, pointer, function, etc... An identifier is a collection of characters which acts as the name of variable, function, array, pointer, structure, etc... In other words, an identifier can be defined as the user-defined name to identify an entity uniquely in the c programming language that name may be of the variable name, function name, array name, pointer name, structure name or a label.

The identifier is a user-defined name of an entity to identify it uniquely during the program execution

Example
in marks; har student Name[30];
Here, marks and studentName are identifiers.

2.6.1 Rules for Creating Identifiers

1. An identifier can contain **letters** (UPPERCASE and lowercase), **numerics** & **underscore** symbol only.
2. An identifier should not start with a numerical value. It can start with a letter or an underscore.

3. We should not use any special symbols in between the identifier even whitespace. However, the only underscore symbol is allowed.
4. Keywords should not be used as identifiers.
5. There is no limit for the length of an identifier. However, the compiler considers the first 31 characters only.
6. An identifier must be unique in its scope.

2.6.2 Rules for Creating Identifiers for better programming

The following are the commonly used rules for creating identifiers for better programming...

1. The identifier must be meaningful to describe the entity.
2. Since starting with an underscore may create conflict with system names, so we avoid starting an identifier with an underscore.
3. We start every identifier with a lowercase letter. If an identifier contains more than one word then the first word starts with a lowercase letter and second word onwards first letter is used as an UPPERCASE letter. We can also use an underscore to separate multiple words in an identifier.

2.7 C DATA TYPES

Data used in c program is classified into different types based on its properties. In the C programming language, a data type can be defined as a set of values with similar characteristics. All the values in a data type have the same properties.

Data types in the c programming language are used to specify what kind of value can be stored in a variable. The memory size and type of the value of a variable are determined by the variable data type. In a c program, each variable or constant or array must have a data type and this data type specifies how much memory is to be allocated and what type of values are to be stored in that variable or constant or array. The formal definition of a data type is as follows...

The Data type is a set of value with predefined characteristics. data types are used to declare variable, constants, arrays, pointers, and functions.

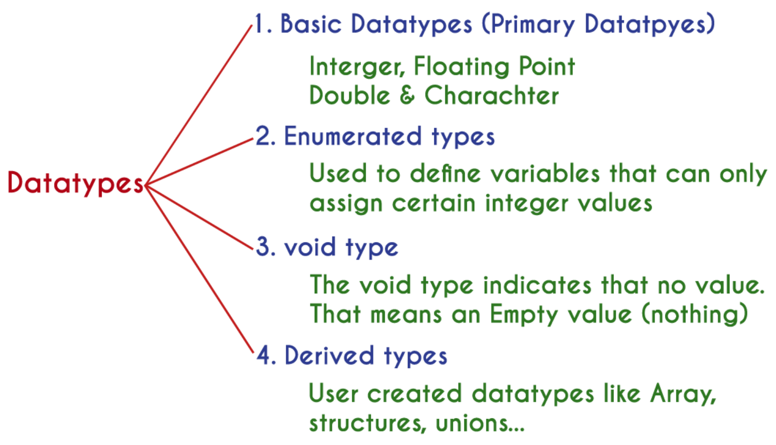
In the **C** programming language, data types are classified as follows...

2 Primary data types (Basic data types OR Predefined data types)

3 Derived data types (Secondary data types OR User-defined data types)

4 Enumeration data types

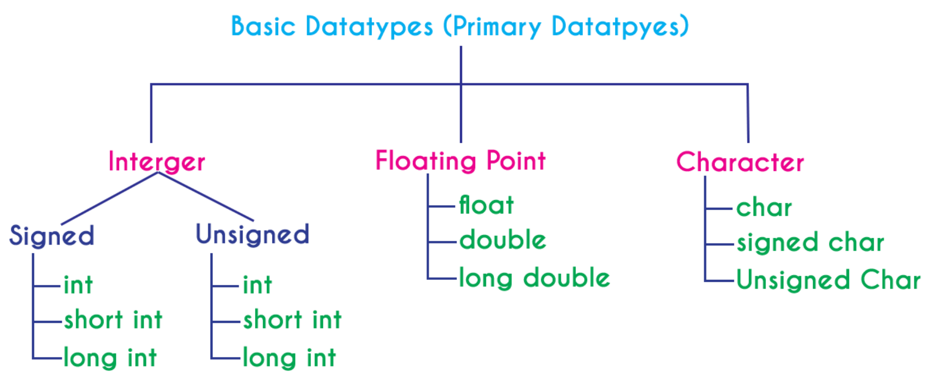
5 Void data type


Datatypes
1. Basic Datatypes (Primary Datatpyes)
Interger, Floating Point
Double & Charachter
2. Enumerated types
Used to define variables that can only
assign certain integer values
3. void type
The void type indicates that no value.
That means an Empty value (nothing)
4. Derived types
User created datatypes like Array,
structures, unions...

2.7.1 Primary data types

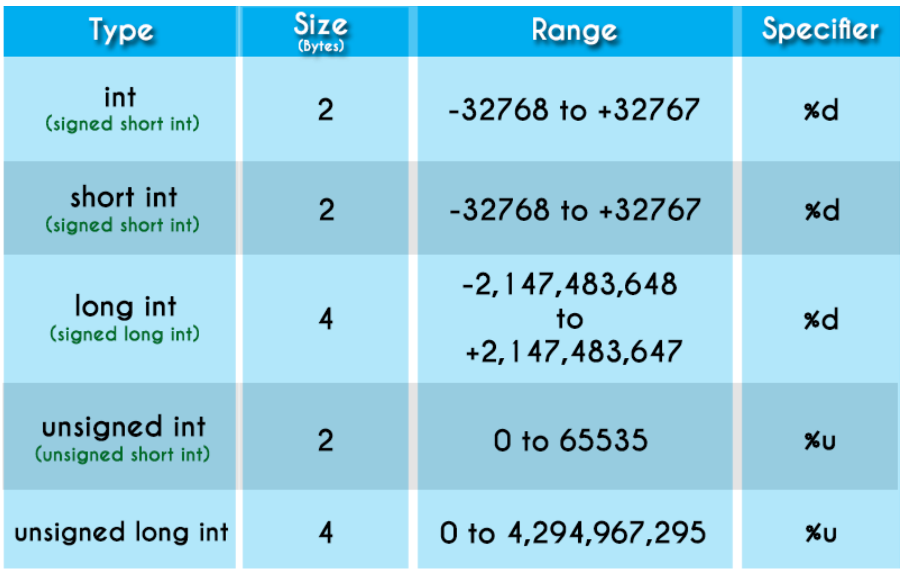
The primary data types in the C programming language are the basic data types. All the primary data types are already defined in the system. Primary data types are also called as Built-In data types. The following are the primary data types in c programming language...

1. Integer data type
2. Floating Point data type
3. Double data type
4. Character data type



Integer Data type

The integer data type is a set of whole numbers. Every integer value does not have the decimal value. We use the keyword "**int**" to represent integer data type in C. We use the keyword int to declare the variables and to specify the return type of a function. The integer data type is used with different type modifiers like short, long, signed and unsigned. The following table provides complete details about the integer data type.



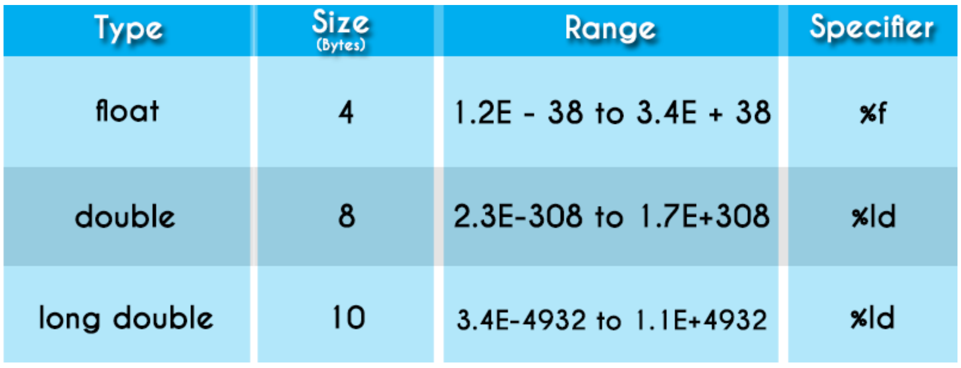
Type	Size (Bytes)	Range	Specifier
int (signed short int)	2	-32768 to +32767	%d
short int (signed short int)	2	-32768 to +32767	%d
long int (signed long int)	4	-2,147,483,648 to +2,147,483,647	%d
unsigned int (unsigned short int)	2	0 to 65535	%u
unsigned long int	4	0 to 4,294,967,295	%u

Floating Point data types

Floating-point data types are a set of numbers with the decimal value. Every floating-point value must contain the decimal value. The floating-point data type has two variants...

- float
- double

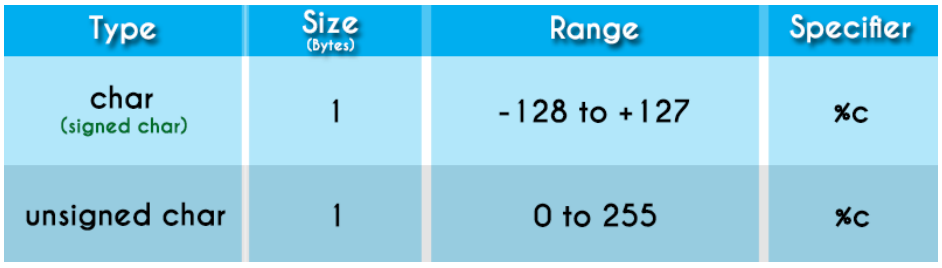
We use the keyword "**float**" to represent floating-point data type and "**double**" to represent double data type in c. Both float and double are similar but they differ in the number of decimal places. The float value contains 6 decimal places whereas double value contains 15 or 19 decimal places. The following table provides complete details about floating-point data types.



Type	Size (Bytes)	Range	Specifier
float	4	1.2E - 38 to 3.4E + 38	%f
double	8	2.3E-308 to 1.7E+308	%ld
long double	10	3.4E-4932 to 1.1E+4932	%ld

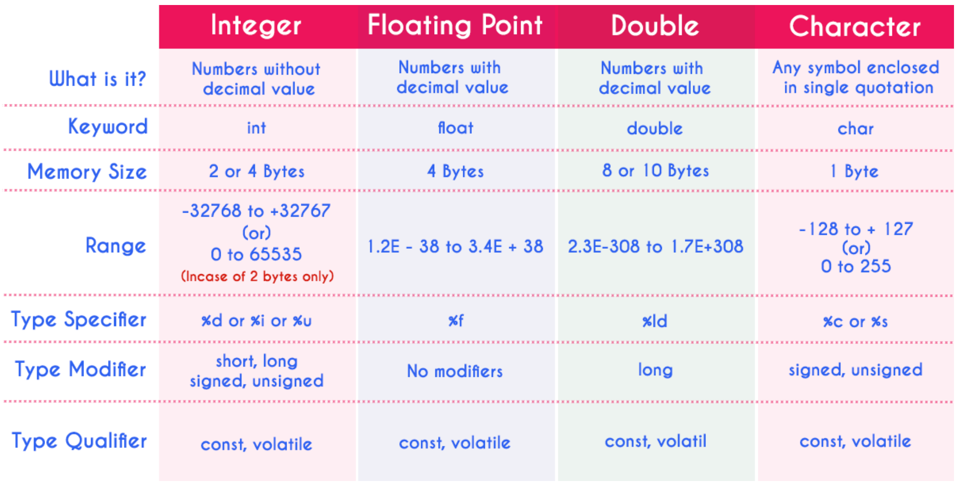
Character data type

The character data type is a set of characters enclosed in single quotations. The following table provides complete details about the character data type.



Type	Size (Bytes)	Range	Specifier
char (signed char)	1	-128 to +127	%c
unsigned char	1	0 to 255	%c

The following table provides complete information about all the data types in c programming language...



	Integer	Floating Point	Double	Character
What is it?	Numbers without decimal value	Numbers with decimal value	Numbers with decimal value	Any symbol enclosed in single quotation
Keyword	int	float	double	char
Memory Size	2 or 4 Bytes	4 Bytes	8 or 10 Bytes	1 Byte
Range	-32768 to +32767 (or) 0 to 65535 (Incase of 2 bytes only)	1.2E - 38 to 3.4E + 38	2.3E-308 to 1.7E+308	-128 to + 127 (or) 0 to 255
Type Specifier	%d or %i or %u	%f	%ld	%c or %s
Type Modifier	short, long signed, unsigned	No modifiers	long	signed, unsigned
Type Qualifier	const, volatile	const, volatile	const, volatil	const, volatile

2.7.2 ENUMERATED DATA TYPE

An enumerated data type is a user-defined data type that consists of integer constants and each integer constant is given a name. The keyword "**enum**" is used to define the enumerated data type.

2.7.3 VOID DATA TYPE

The void data type means nothing or no value. Generally, the void is used to specify a function which does not return any value. We also use the void data type to specify empty parameters of a function.

2.7.4 DERIVED DATA TYPES

Derived data types are user-defined data types. The derived data types are also called as user-defined data types or secondary data types. In the c programming language, the derived data types are created using the following concepts...

- Arrays
- Structures
- Unions
- Enumeration

2.7 LET US SUM UP

We understood the basic and essential elements that will be utilized further to write a program in C language.

2.8 KEYWORDS

Guideline - a general rule, principle, or piece of advice.

Keyword- a word which acts as the key to a cipher or code.

Decimal Value - A representation of a real number using the base ten and **decimal** notation, such as 201.4, 3.89

2.9 QUESTIONS FOR REVIEW

1. State General rules for any C program

2. What are Identifiers and explain the related rules for creating identifiers.

3.Explain the classification of data types in C.

2.10 SUGGESTED READINGS AND REFERENCES

1. Fundamentals of Real Analysis, S K. Berberian, Springer.
2. Measure Theory and Integration, G. De Barra, New Age International Publ.
3. Real Analysis, H. L. Royden.
4. Principles of Mathematical Analysis, W. Rudin.
5. Lectures on Real Analysis, J. Yeh, World Sci.
6. R. G. Bartle, The Elements of Integration, John Wiley & Sons, Inc. New York, 1966

2.11 ANSWERS TO CHECK YOUR PROGRESS

1. Provide definition- 2.2

2. Provide explanation –2.3

3.Provide definition – 2.4

4. Provide definition and few examples– 2.5

UNIT-3 ELEMENTS OF C -I

STRUCTURE

3.0 OBJECTIVES

Understand the different elements of C language

C Variable

C Constant

C Storage Class

C Output Function

3.1 INTRODUCTION

Before going in depth, we need to understand the basic elements of C language. It is very necessary to know these elements because it helps you to write C program.

3.2 C VARIABLES

Variables in a C programming language are the named memory locations where the user can store different values of the same datatype during the program execution. That means a variable is a name given to a memory location in which we can store different values of the same data type. In other words, a variable can be defined as a storage container to hold values of the same datatype during the program execution. The formal definition of a data type is as follows...
Variable is a name given to a memory location where we can store different values of the same datatype during the program execution.

Every variable in C programming language must be declared in the declaration section before it is used. Every variable must have a datatype that determines the range and type of values be stored and the size of the memory to be allocated.

A variable name may contain letters, digits and underscore symbol. The following are the rules to specify a variable name...

1. Variable name should not start with a digit.
2. Keywords should not be used as variable names.
3. A variable name should not contain any special symbols except underscore(_).
4. A variable name can be of any length but compiler considers only the first 31 characters of the variable name.

3.2.1 DECLARATION OF VARIABLE

Declaration of a variable tells the compiler to allocate the required amount of memory with the specified variable name and allows only specified datatype values into that memory location. In C programming language, the declaration can be performed either before the function as global variables or inside any block or function. But it must be at the beginning of block or function.

Declaration Syntax

```
datatype variableName;
```

Example

int number;

The above declaration tells to the compiler that allocates **2 bytes** of memory with the name **number** and allows only integer values into that memory location.

3.3 C CONSTANTS

In C programming language, a constant is similar to the variable but the constant hold only one value during the program execution. That means, once a value is assigned to the constant, that value can't be changed during the program execution. Once the value is assigned to the constant,

it is fixed throughout the program. A constant can be defined as follows...
A constant is a named memory location which holds only one value throughout the program execution.

In C programming language, a constant can be of any data type like integer, floating-point, character, string and double, etc.,

3.3.1 INTEGER CONSTANTS

An integer constant can be a decimal integer or octal integer or hexadecimal integer. A decimal integer value is specified as direct integer value whereas octal integer value is prefixed with 'o' and hexadecimal value is prefixed with 'OX'.
An integer constant can also be unsigned type of integer constant or long type of integer constant. Unsigned integer constant value is suffixed with 'u' and long integer constant value is suffixed with 'l' whereas unsigned long integer constant value is suffixed with 'ul'.

Example

125 -----> Decimal Integer Constant

O76 -----> Octal Integer Constant

OX3A -----> Hexa Decimal Integer Constant

50u -----> Unsigned Integer Constant

30l -----> Long Integer Constant

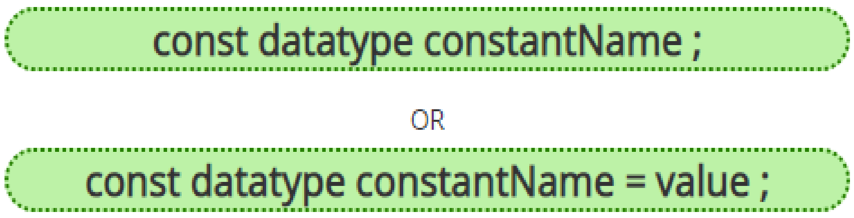
100ul -----> Unsigned Long Integer Constant

3.3.2 FLOATING POINT CONSTANTS

A floating-point constant must contain both integer and decimal parts. Sometimes it may also contain the exponent part. When a floating-point constant is represented in exponent form, the value must be suffixed with 'e' or 'E'.

Example

The floating-point value **3.14** is represented as **3E-14** in exponent form.



```
const datatype constantName ;
OR
const datatype constantName = value ;
```

3.3.3 CHARACTER CONSTANTS

A character constant is a symbol enclosed in single quotation. A character constant has a maximum length of one character.

Example

'A'

'2'

'+'

In the C programming language, there are some predefined character constants called escape sequences. Every escape sequence has its own special functionality and every escape sequence is prefixed with '\' symbol. These escape sequences are used in output function called 'printf()'.

3.3.4 STRING CONSTANTS

A string constant is a collection of characters, digits, special symbols and escape sequences that are enclosed in double quotations.

We define string constant in a single line as follows...

"This is btechsmartclass"

We can define string constant using multiple lines as follows...

" This\

is

btechsmartclass "

We can also define string constant by separating it with white space as follows...

"This" "is" "btechsmartclass"

All the above three defines the same string constant.

3.3.5 CREATING CONSTANTS IN C

In a C programming language, constants can be created using two concepts...

1. **Using the 'const' keyword**
2. **Using '#define' preprocessor**

USING THE 'CONST' KEYWORD

We create a constant of any datatype using 'const' keyword. To create a constant, we prefix the variable declaration with 'const' keyword.
The general syntax for creating constant using 'const' keyword is as follows...

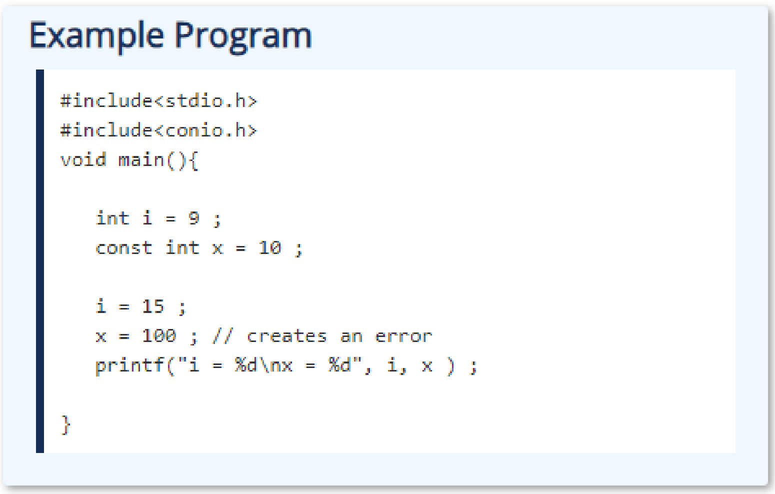
const datatype constantName ;

OR

const datatype constantName = value ;

Example
const int x = 10 ;

Here, 'x' is a integer constant with fixed value 10.



Example Program

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

   int i = 9 ;
   const int x = 10 ;

   i = 15 ;
   x = 100 ; // creates an error
   printf("i = %d\nx = %d", i, x ) ;

}
```

The above program gives an **error** because we are trying to change the constant variable value (x = 100).

USING '#DEFINE' PREPROCESSOR

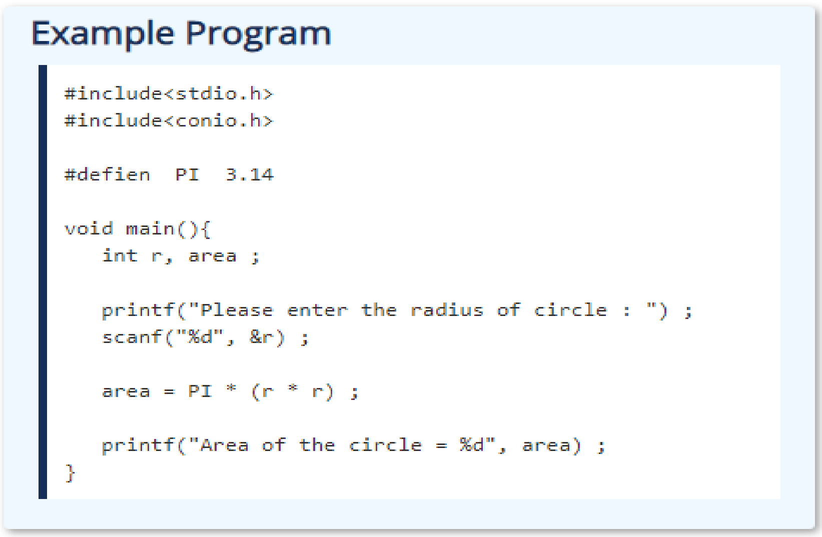
We can also create constants using '#define' preprocessor directive. When we create constant using this preprocessor directive it must be defined at the beginning of the program (because all the preprocessor directives must be written before the global declaration). We use the following syntax to create constant using '#define' preprocessor directive...

```
#define CONSTANTNAME value
```

Example

#define PI 3.14

Here, **PI** is a constant with value **3.14**


Example Program

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

#defien  PI  3.14

void main(){
   int r, area ;

   printf("Please enter the radius of circle : ") ;
   scanf("%d", &r) ;

   area = PI * (r * r) ;

   printf("Area of the circle = %d", area) ;
}
```

Check your Progress-1

1. State the rules to specify the variable name

____

____

____

2. Define constant with examples

____

____

____

3.4 C STORAGE CLASSES

In C programming language, storage classes are used to define things like **storage location** (whether RAM or REGISTER), **scope**, **lifetime** and the **default value** of a variable.

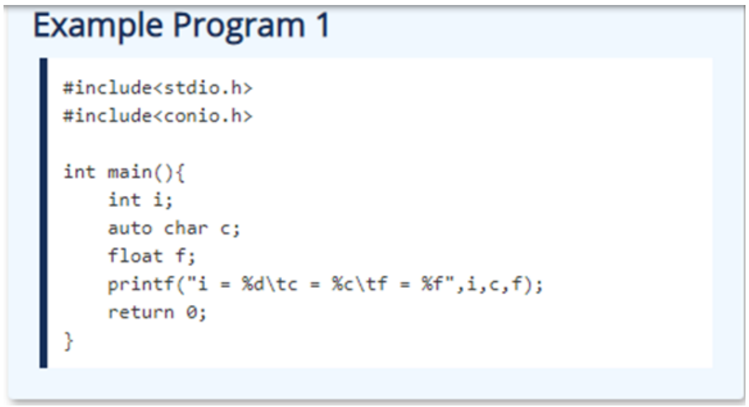
In the **C** programming language, the memory of variables is allocated either in computer memory (RAM) or CPU Registers. The allocation of memory depends on storage classes.

In **C** programming language, there are FOUR storage classes and they are as follows...

1. **Auto storage class**
2. **Extern storage class**
3. **Static storage class**
4. **Register storage class**

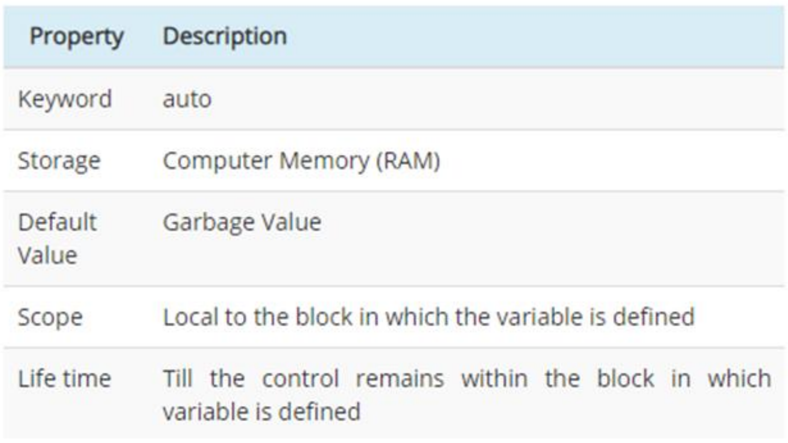
3.4.1 Auto Storage Class

The default storage class of all local variables (variables declared inside block or function) is auto storage class. Variable of auto storage class has the following properties...

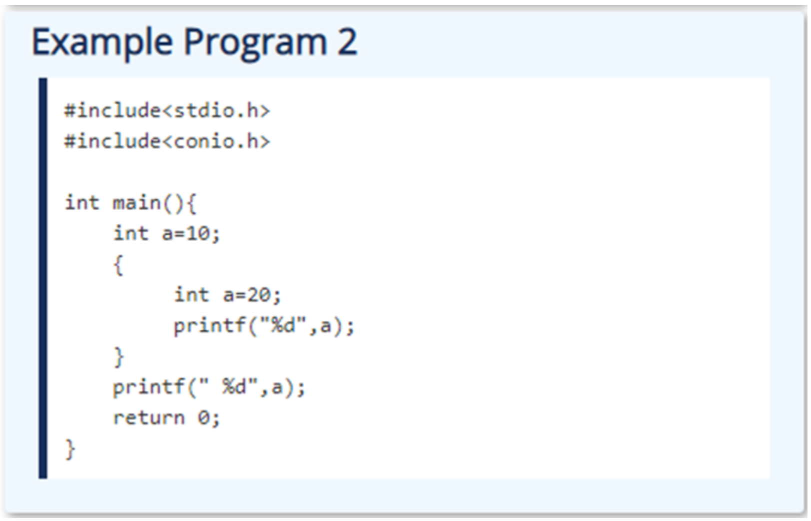


Example Program 1

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

int main(){
    int i;
    auto char c;
    float f;
    printf("i = %d\tc = %c\tf = %f",i,c,f);
    return 0;
}
```

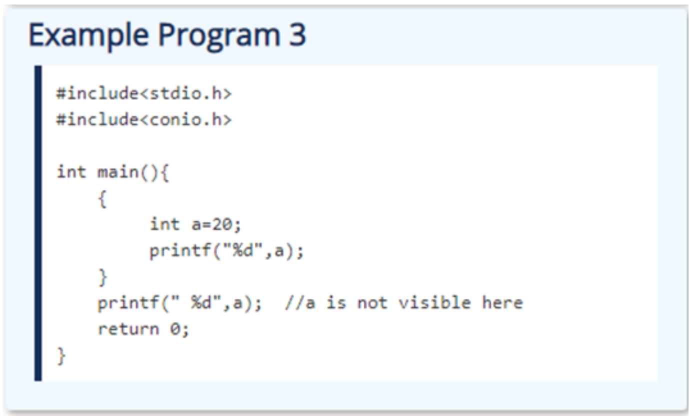


Property	Description
Keyword	auto
Storage	Computer Memory (RAM)
Default Value	Garbage Value
Scope	Local to the block in which the variable is defined
Life time	Till the control remains within the block in which variable is defined



Example Program 2

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

int main(){
    int a=10;
    {
         int a=20;
         printf("%d",a);
    }
    printf(" %d",a);
    return 0;
}
```

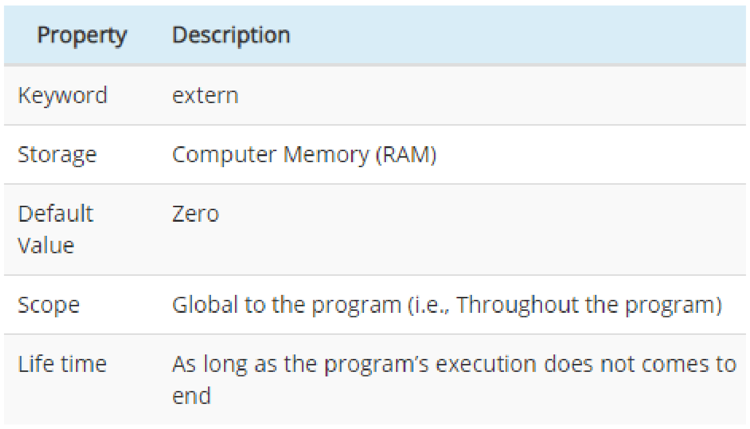


Example Program 3

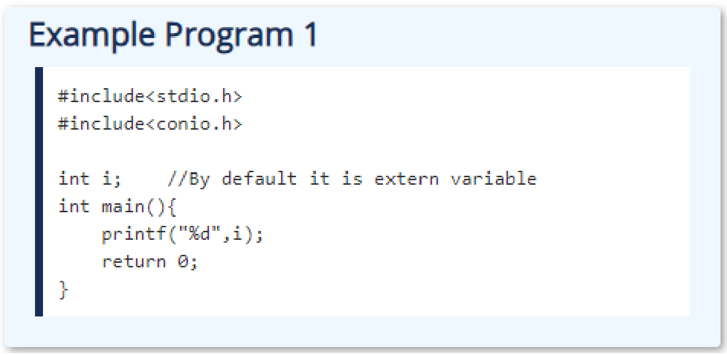
```
#include<stdio.h>
#include<conio.h>

int main(){
    {
         int a=20;
         printf("%d",a);
    }
    printf(" %d",a);  //a is not visible here
    return 0;
}
```

3.4.2 EXTERNAL STORAGE CLASS

The default storage class of all global variables (variables declared outside function) is external storage class. Variable of external storage class has the following properties...

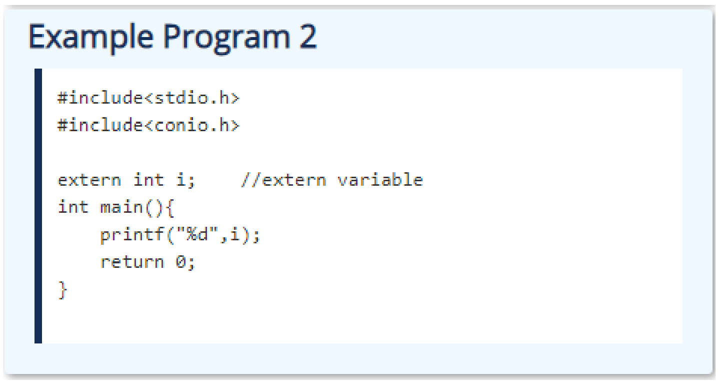


Property	Description
Keyword	extern
Storage	Computer Memory (RAM)
Default Value	Zero
Scope	Global to the program (i.e., Throughout the program)
Life time	As long as the program's execution does not comes to end



Example Program 1

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

int i;    //By default it is extern variable
int main(){
    printf("%d",i);
    return 0;
}
```



Example Program 2

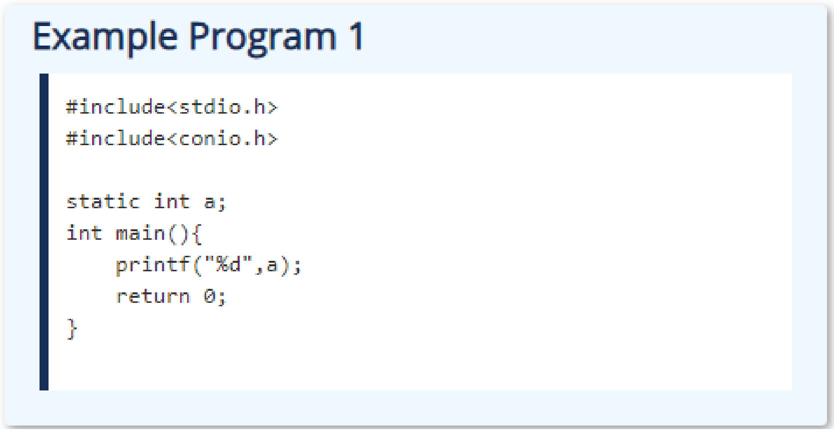
```
#include<stdio.h>
#include<conio.h>

extern int i;    //extern variable
int main(){
    printf("%d",i);
    return 0;
}
```

3.4.3 STATIC STORAGE CLASS

The static storage class is used to create variables that hold value beyond its scope until the end of the program. The static variable allows to initialize only once and can be modified any number of times. Variable of static storage class has the following properties...

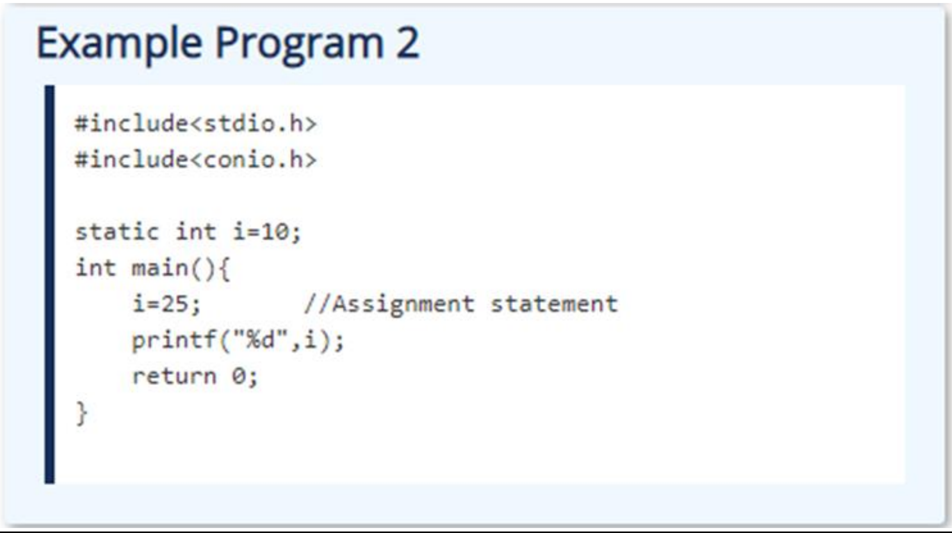
Property	Description
Keyword	static
Storage	Computer Memory (RAM)

Default Value	Zero
Scope	Local to the block in which the variable is defined
Life time	The value of the persists between different function calls (i.e., Initialization is done only once)



Example Program 1

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

static int a;
int main(){
    printf("%d",a);
    return 0;
}
```



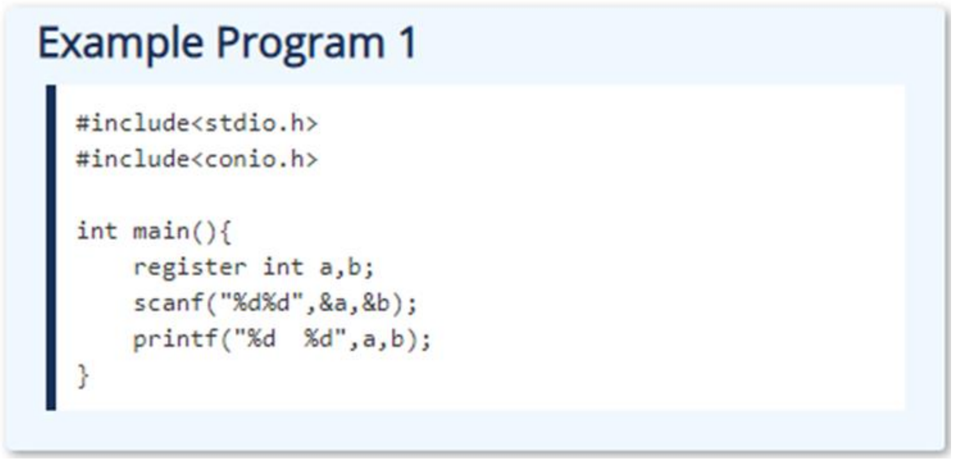
Example Program 2

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

static int i=10;
int main(){
    i=25;       //Assignment statement
    printf("%d",i);
    return 0;
}
```

3.4.4 REGISTER STORAGE CLASS

The register storage class is used to specify the memory of the variable that has to be allocated in CPU Registers. The memory of the register variable is allocated in CPU Register but not in Computer memory (RAM). The register variables enable faster accessibility compared to other storage class variables. As the number of registers inside the CPU is very less we can use very less number of register variables. Variable of register storage class has the following properties...

Property	Description
Keyword	register
Storage	CPU Register
Default Value	Garbage Value
Scope	Local to the block in which the variable is defined
Life time	Till the control remains within the block in which variable is defined


Example Program 1

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

int main(){
    register int a,b;
    scanf("%d%d",&a,&b);
    printf("%d  %d",a,b);
}
```

The following table provides detailed properties of all storage classes...

Storage Class	Keyword	Memory Location	Default Value	Scope	Life Time
Automatic	auto	Computer Memory (RAM)	Garbage Value	Local to the block in which the variable	Till the control remains within the block in

				has defined	which variable is defined
External	extern	Computer Memory (RAM)	Zero	Global to the program (i.e., Throughout the program)	As long as the program's execution does not come to end
Static	static	Computer Memory (RAM)	Zero	Local to the block in which the variable has defined	The value of the persists between different function calls (i.e., Initialization is done only once)
Register	register	CPU Register	Garbage Value	Local to the block in which the variable has defined	Till the control remains within the block in which variable is defined

3.5 C OUTPUT FUNCTIONS

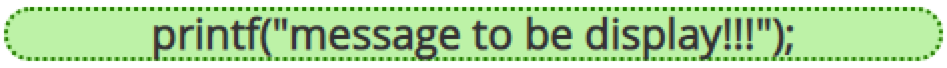
C programming language provides built-in functions to perform output operation. The output operations are used to display data on user screen (output screen) or printer or any file. The c programming language provides the following built-in output functions...

1. **printf()**
2. **putchar()**
3. **puts()**
4. **fprintf()**

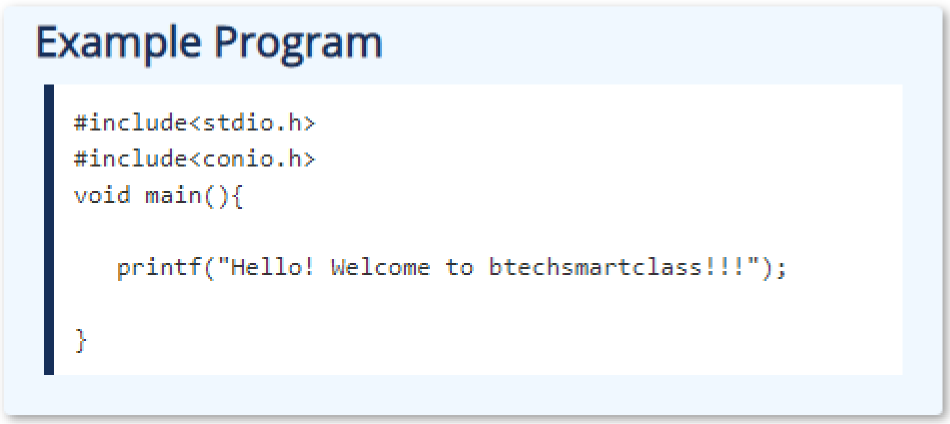
3.5.1 printf() function

The printf() function is used to print string or data values or a combination of string and data values on the output screen (User screen). The printf() function is built-in function defined in a header file called "**stdio.h**". When we want to use printf() function in our program we need to include the respective header file (stdio.h) using the **#include** statement. The printf() function has the following syntax...

Syntax:



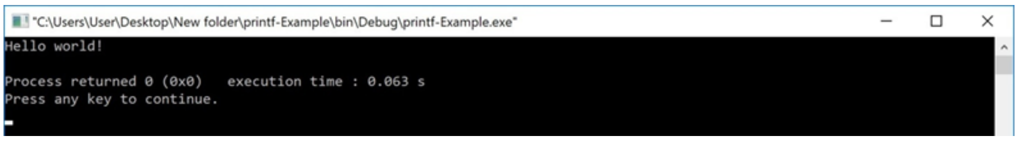
```
printf("message to be display!!!");
```



Example Program

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

   printf("Hello! Welcome to btechsmartclass!!!");

}
```

Output:



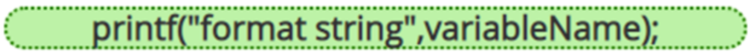
"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

```
Hello world!

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

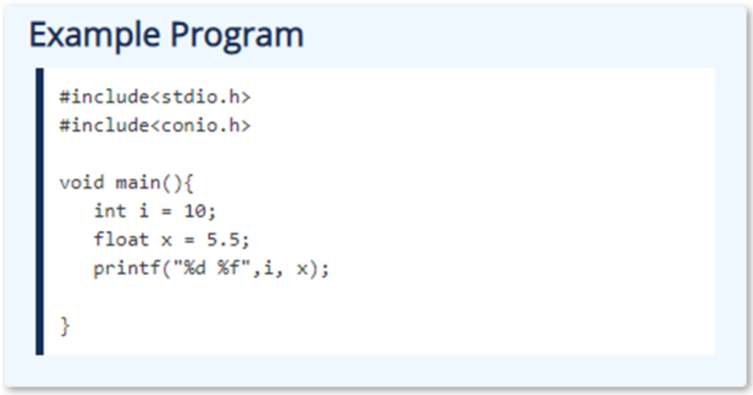
In the above example program, we used the printf() function to print a string on to the output screen.

The printf() function is also used to display data values. When we want to display data values we use **format string** of the data value to be displayed.

Syntax:



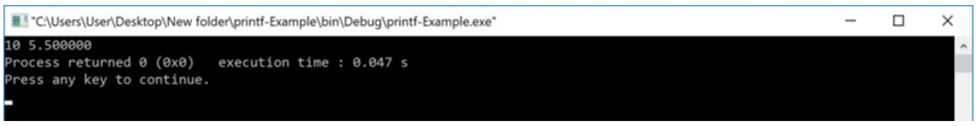
```
printf("format string",variableName);
```



Example Program

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

void main(){
   int i = 10;
   float x = 5.5;
   printf("%d %f",i, x);

}
```

Output:



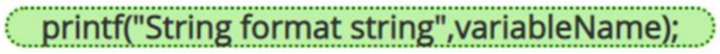
"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

```
10 5.500000
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

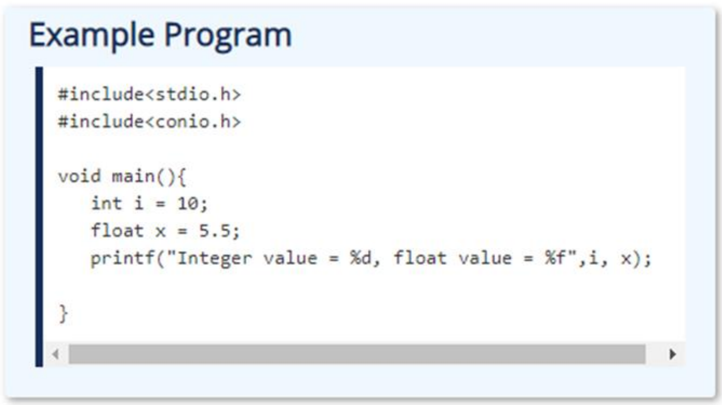
In the above example program, we used the printf() function to print data values of variables i and x on to the output screen. Here i is a an integer variable so we have used format string %d and x is a float variable so we have used format string %f.

The print() function can also be used to display string along with data values.

Syntax:



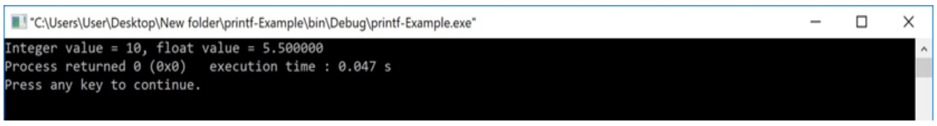
```
printf("String format string",variableName);
```



Example Program

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

void main(){
   int i = 10;
   float x = 5.5;
   printf("Integer value = %d, float value = %f",i, x);

}
```

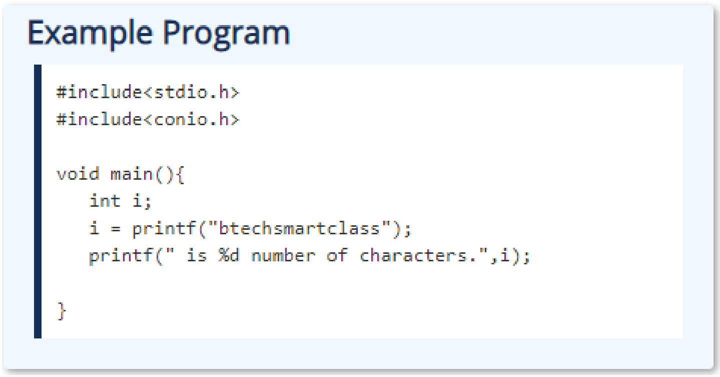
Output:



"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

```
Integer value = 10, float value = 5.500000
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

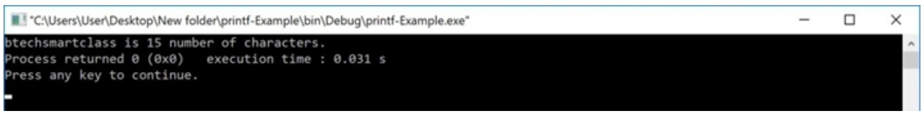
In the above program, we are displaying string along with data values. Every function in the C programming language must have a return value. The printf() function also have an integer as a return value. The printf() function returns an integer value equivalent to the total number of characters it has printed.



Example Program

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

void main(){
   int i;
   i = printf("btechsmartclass");
   printf(" is %d number of characters.",i);

}
```

Output:



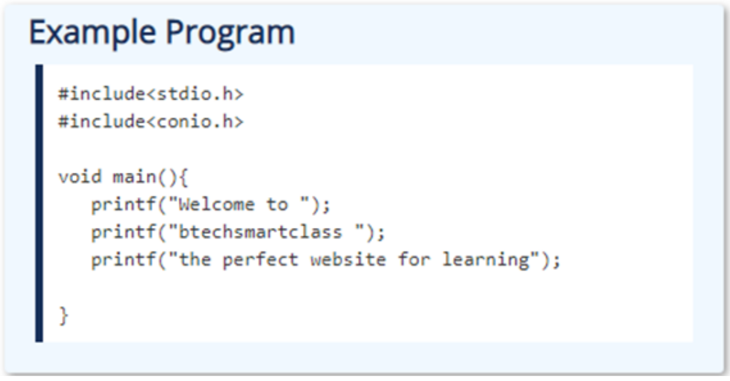
"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

```
btechsmartclass is 15 number of characters.
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

In the above program, first printf() function printing "btechsmartclass" which is of 15 characters. So it returns integer value 15 to the variable "i". The value of "i" is printed in the second printf() function.

3.5.2 Formatted printf() function

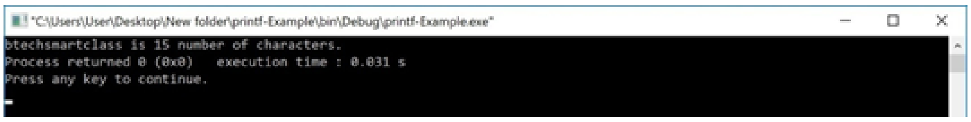
Generally, when we write multiple printf() statements the result is displayed in a single line because the printf() function displays the output in a single line. Consider the following example program...



Example Program

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

void main(){
   printf("Welcome to ");
   printf("btechsmartclass ");
   printf("the perfect website for learning");

}
```

Output



"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

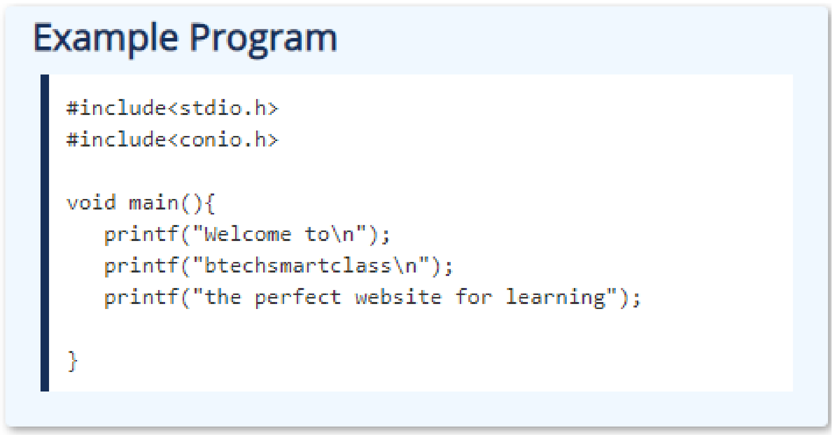
```
btechsmartclass is 15 number of characters.
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

In the above program, there are 3 printf() statements written in different lines but the output is displayed in single line only.

To display the output in different lines or as we wish, we use some special characters called **escape sequences**. Escape sequences are special characters with special functionality used in printf() function to format the output according to the user requirement. In the C programming language, we have the following escape sequences...

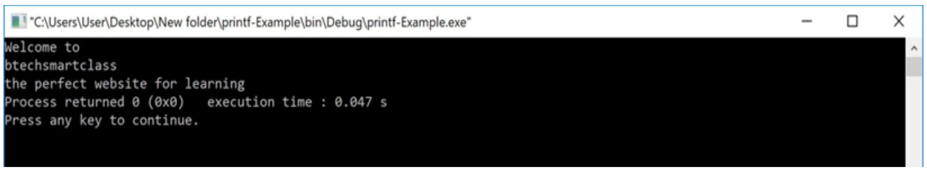
Escape sequence	**Meaning**
\n	Moves the cursor to New Line
\t	Inserts Horizontal Tab (5 characters space)
\v	Inserts Vertical Tab (5 lines space)
\a	Beep sound
\b	Backspace (removes the previous character from its current position)
\\	Inserts Backward slash symbol
\?	Inserts Question mark symbol
\'	Inserts Single quotation mark symbol
\"	Inserts Double quotation mark symbol

Consider the following example program...



Example Program

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

void main(){
   printf("Welcome to\n");
   printf("btechsmartclass\n");
   printf("the perfect website for learning");

}
```

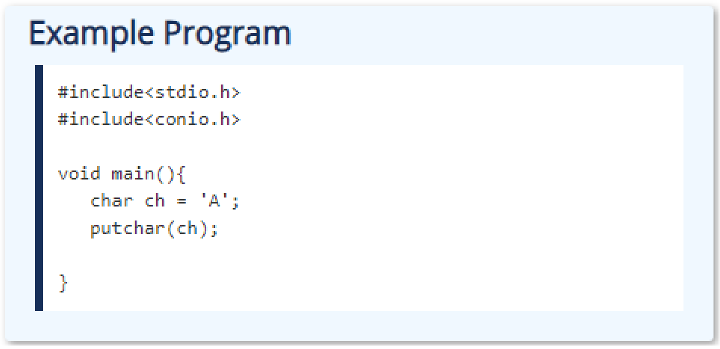


"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

```
Welcome to
btechsmartclass
the perfect website for learning
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

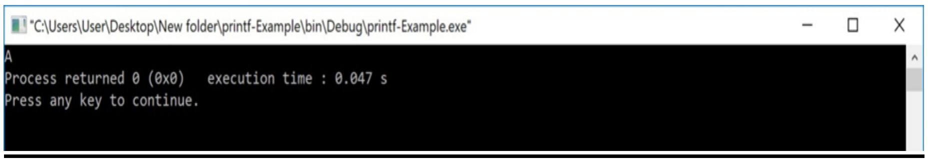
3.5.3 putchar() function

The putchar() function is used to display a single character on the output screen. The putchar() functions prints the character which is passed as a parameter to it and returns the same character as a return value. This function is used to print only a single character. To print multiple characters we need to write multiple times or use a looping statement. Consider the following example program...



Example Program

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

void main(){
   char ch = 'A';
   putchar(ch);

}
```

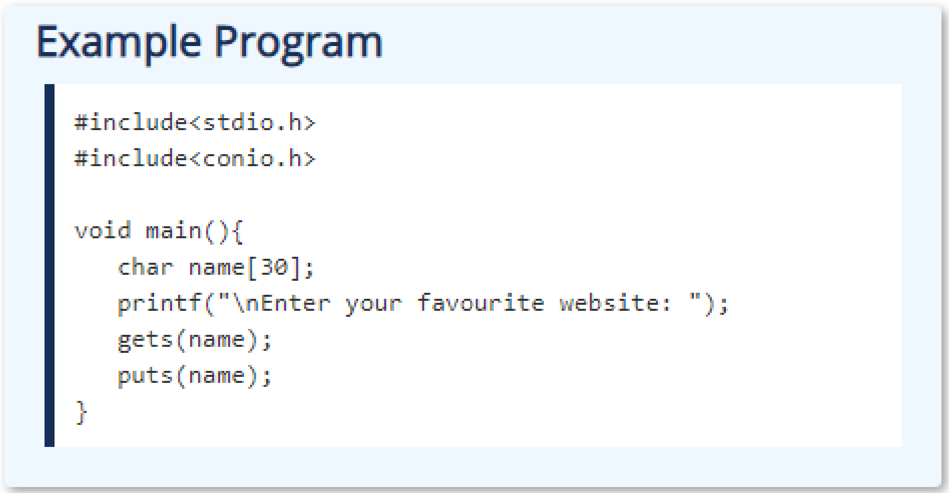


"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

```
A
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

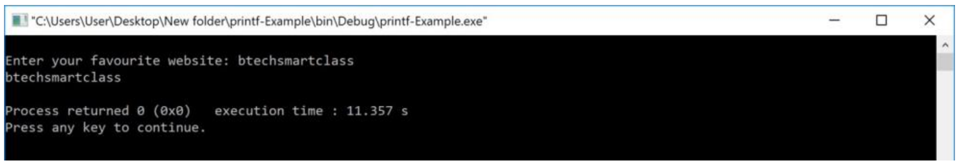
3.5.4 puts() function

The puts() function is used to display a string on the output screen. The puts() functions prints a string or sequence of characters till the newline. Consider the following example program...



Example Program

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

void main(){
   char name[30];
   printf("\nEnter your favourite website: ");
   gets(name);
   puts(name);
}
```



"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"

```
Enter your favourite website: btechsmartclass
btechsmartclass

Process returned 0 (0x0)   execution time : 11.357 s
Press any key to continue.
```

3.5.5 fprintf() function

The fprintf() function is used with the concept of files. The fprintf() function is used to print a line into the file. When you want to use fprintf() function the file must be opened in writting mode.

Check your Progress-2

3.What do you understand by External Storage Class?

____

____

____

4. What is formatted printf function?

3.6 LET US SUM UP

We came across various elements used in C language programming and also understood clearly how to utilize these elements while writing a program.

3.7 KEYWORDS

Memory location - A byte, word or other small unit of storage space in a computer's main **memory** that is identified by its starting **address** (and size)**Keyword**- a word which acts as the key to a cipher or code.

Initialize - set to the value or put in the condition appropriate to the start of an operation.

Specify - state a fact or requirement clearly and precisely.

Escape sequences- are special characters with special functionality used in printf() function to format the output according to the user requirement

3.8 QUESTIONS FOR REVIEW

1. Explain Declaration of variable

2. Discuss creating constants in C.

3. Explain auto storage class with properties and example

4. Provide detailed properties of all storage classes

3.9 SUGGESTED READINGS AND REFERENCES

1. Fundamentals of Real Analysis, S K. Berberian, Springer.

2. Measure Theory and Integration, G. De Barra, New Age International Publ.
3. Real Analysis, H. L. Royden.
4. Principles of Mathematical Analysis, W. Rudin.
5. Lectures on Real Analysis, J. Yeh, World Sci.
6. R. G. Bartle, The Elements of Integration, John Wiley & Sons, Inc. New York, 1966

3.10 ANSWERS TO CHECK YOUR PROGRESS

1. Provide rules- 3.2

2. Provide definition & example –3.3

3.Provide definition – 2.4

4. Provide definition, syntax and example– 3.5.2

UNIT-4 ELEMENTS OF C -II

STRUCTURE

4.0 OBJECTIVES

Understands the **C** Input Functions

Comprehend the different operators in C and their applications

4.1 INTRODUCTION

C programming language provides built-in functions to perform input operations to read any given input and to display data on screen when there is a need to output the result. C programming language supports a rich set of operators.

4.2 C INPUT FUNCTIONS

The input operations are used to read user values (input) from the keyboard. The C programming language provides the following built-in input functions.

1 scanf()

2 getchar()

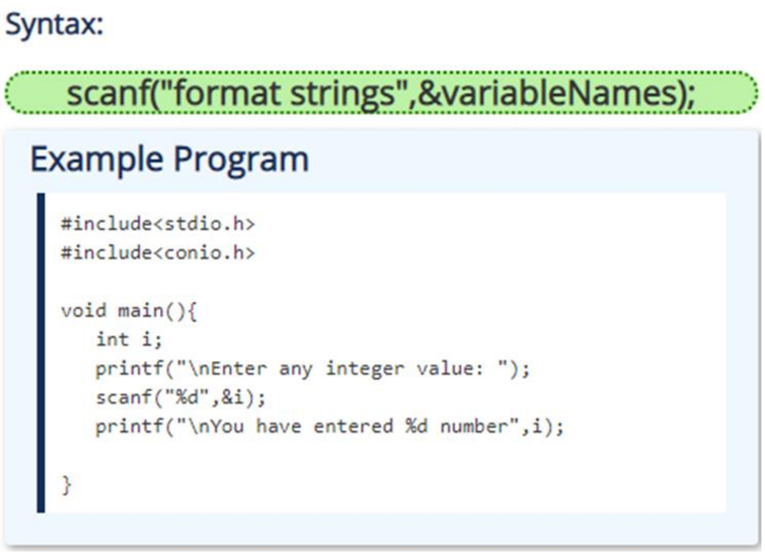
3 getch()

4 gets()

5 fscanf()

4.2.1 scanf() function

The scanf() function is used to read multiple data values of different data types from the keyboard. The scanf() function is built-in function defined in a header file called "**stdio.h**". When we want to use scanf() function in our program, we need to include the respective header file (stdio.h) using **#include** statement. The scanf() function has the following syntax...



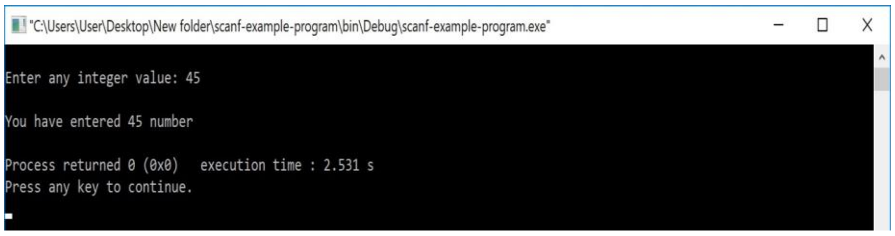
Syntax:

```
scanf("format strings",&variableNames);
```

Example Program

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

void main(){
   int i;
   printf("\nEnter any integer value: ");
   scanf("%d",&i);
   printf("\nYou have entered %d number",i);

}
```

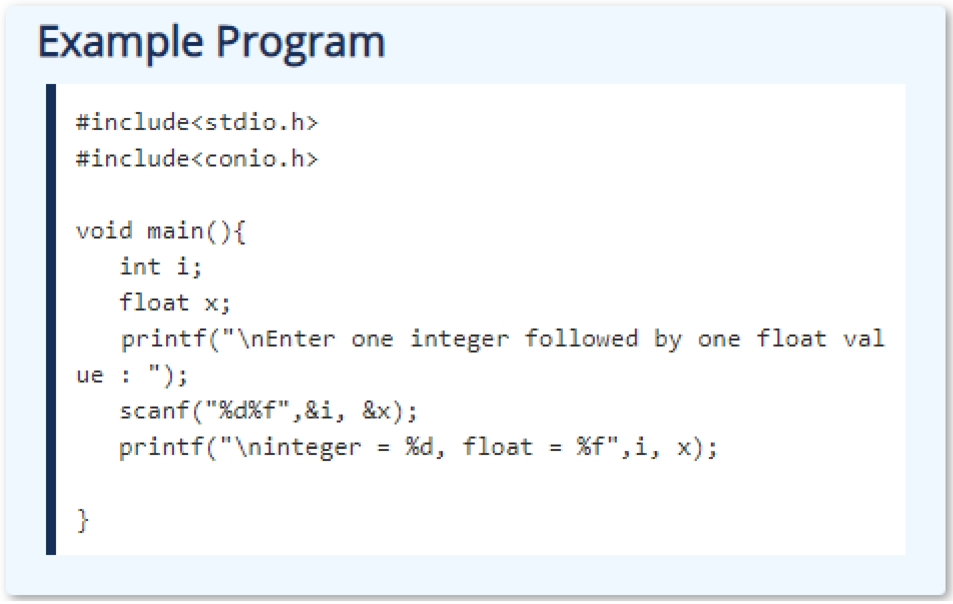
Output:



"C:\Users\User\Desktop\New folder\scanf-example-program\bin\Debug\scanf-example-program.exe"

```
Enter any integer value: 45

You have entered 45 number

Process returned 0 (0x0)   execution time : 2.531 s
Press any key to continue.
```

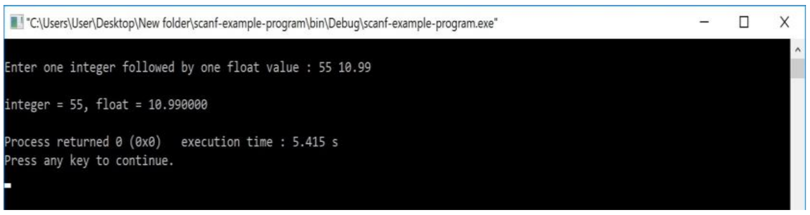
In the above example program, we used the scanf() function to read an integer value from the keyboard and store it into variable 'i'.
The scanf function also used to read multiple data values of different or the same data types. Consider the following example program...



Example Program

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

void main(){
   int i;
   float x;
   printf("\nEnter one integer followed by one float val
ue : ");
   scanf("%d%f",&i, &x);
   printf("\ninteger = %d, float = %f",i, x);

}
```

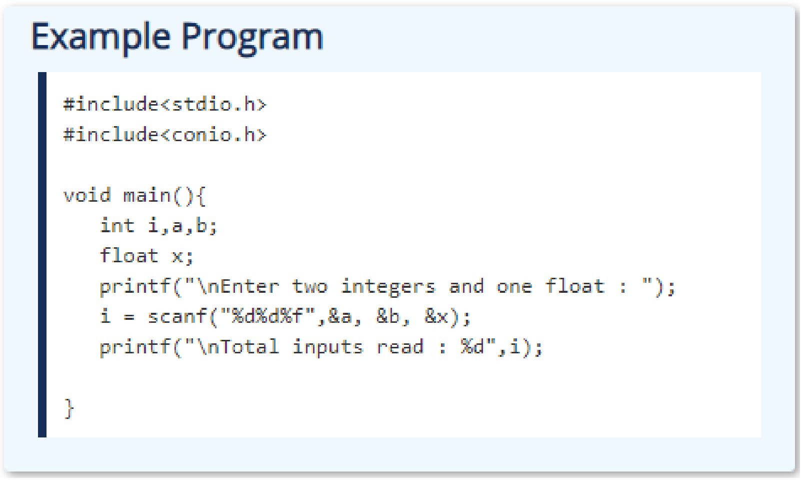
Output:



```
"C:\Users\User\Desktop\New folder\scanf-example-program\bin\Debug\scanf-example-program.exe"

Enter one integer followed by one float value : 55 10.99

integer = 55, float = 10.990000

Process returned 0 (0x0)   execution time : 5.415 s
Press any key to continue.
```

In the above example program, we used the scanf() function to read one integer value and one float value from the keyboard. Here **'i'** is an integer variable so we have used format string %d, and **'x'** is a float variable so we have used format string %f.

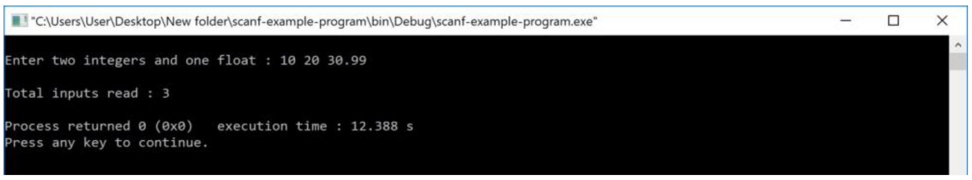
The scanf() function returns an integer value equal to the total number of input values read using scanf function.



Example Program

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

void main(){
   int i,a,b;
   float x;
   printf("\nEnter two integers and one float : ");
   i = scanf("%d%d%f",&a, &b, &x);
   printf("\nTotal inputs read : %d",i);

}
```

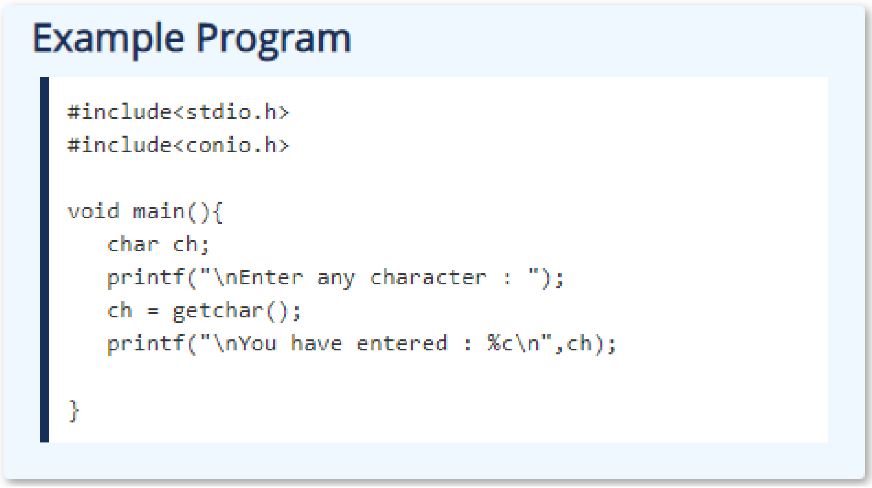
Output:



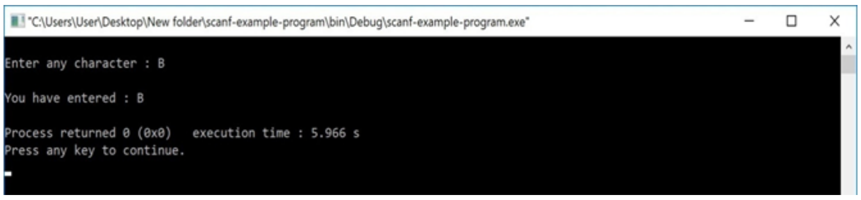
```
"C:\Users\User\Desktop\New folder\scanf-example-program\bin\Debug\scanf-example-program.exe"
Enter two integers and one float : 10 20 30.99

Total inputs read : 3

Process returned 0 (0x0)   execution time : 12.388 s
Press any key to continue.
```

4.2.2 getchar() function

The getchar() function is used to read a character from the keyboard and return it to the program. This function is used to read a single character. To read multiple characters we need to write multiple times or use a looping statement. Consider the following example program...



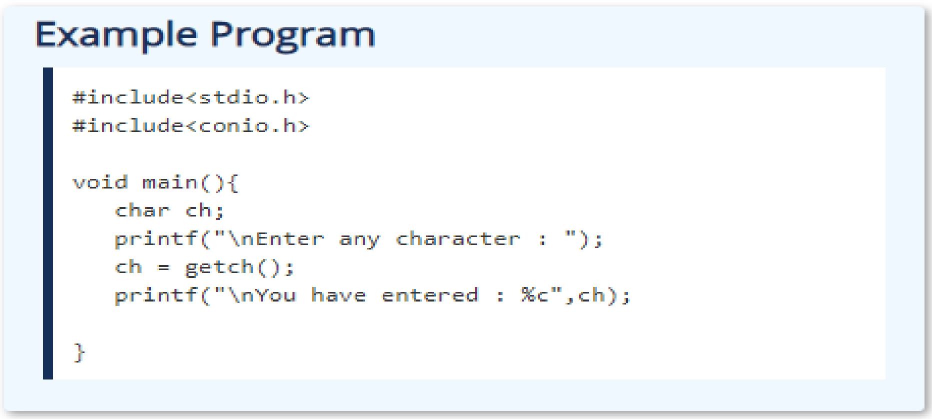
```
Example Program

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

void main(){
   char ch;
   printf("\nEnter any character : ");
   ch = getchar();
   printf("\nYou have entered : %c\n",ch);

}
```



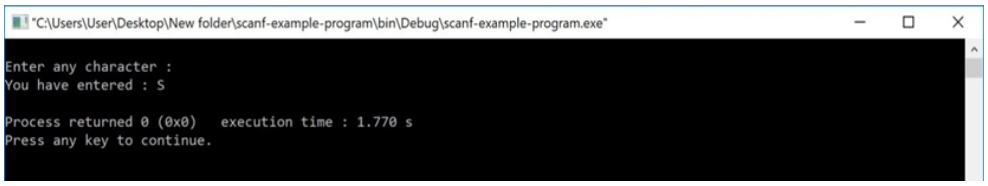
```
"C:\Users\User\Desktop\New folder\scanf-example-program\bin\Debug\scanf-example-program.exe"

Enter any character : B

You have entered : B

Process returned 0 (0x0)   execution time : 5.966 s
Press any key to continue.
```

4.2.3 getch() function

The getch() function is similar to getchar function. The getch() function is used to read a character from the keyboard and return it to the program. This function is used to read a single character. To read multiple characters we need to write multiple times or use a looping statement. Consider the following example program...



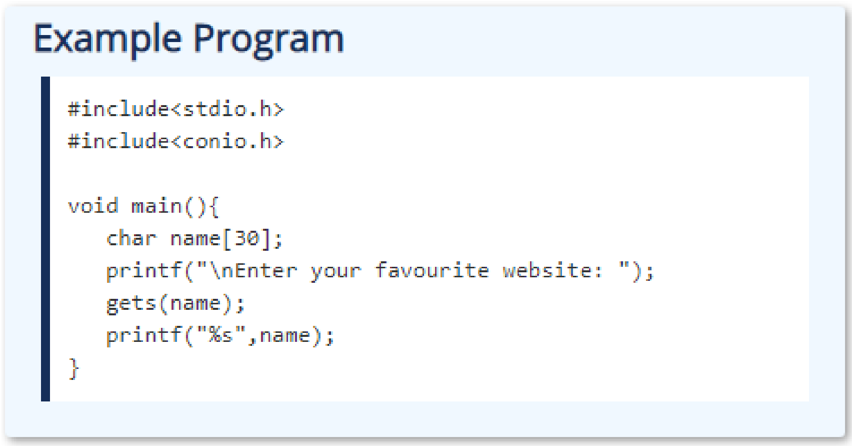
```
Example Program

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

void main(){
   char ch;
   printf("\nEnter any character : ");
   ch = getch();
   printf("\nYou have entered : %c",ch);

}
```



```
"C:\Users\User\Desktop\New folder\scanf-example-program\bin\Debug\scanf-example-program.exe"

Enter any character :
You have entered : S

Process returned 0 (0x0)   execution time : 1.770 s
Press any key to continue.
```

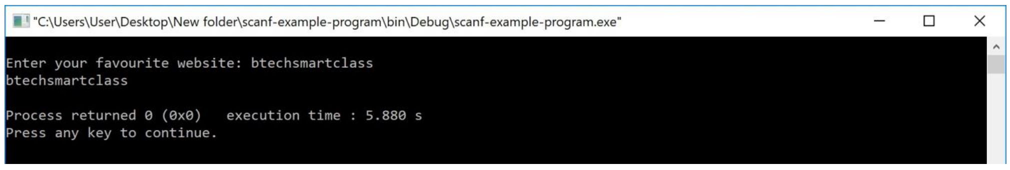
4.2.4 gets() function

The gets() function is used to read a line of string and stores it into a character array. The gets() function reads a line of string or sequence of characters till a newline symbol enters. Consider the following example program...



Example Program

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

void main(){
   char name[30];
   printf("\nEnter your favourite website: ");
   gets(name);
   printf("%s",name);
}
```



```
"C:\Users\User\Desktop\New folder\scanf-example-program\bin\Debug\scanf-example-program.exe"

Enter your favourite website: btechsmartclass
btechsmartclass

Process returned 0 (0x0)   execution time : 5.880 s
Press any key to continue.
```

4.2.5 fscanf() function

The fscanf() function is used with the concept of files. The fscanf() function is used to read data values from a file. When you want to use fscanf() function the file must be opened in reading mode.

Check your Progress-1

1. What do you understand by getchar() function?

____

____

____

2. Define fscanf() function

4.3 C OPERATORS

An operator is a symbol used to perform arithmetic and logical operations in a program. That means an operator is a special symbol that tells the compiler to perform mathematical or logical operations. C programming language supports a rich set of operators that are classified as follows.

1 **Arithmetic Operators**
2 **Relational Operators**
3 **Logical Operators**
4 **Increment & Decrement Operators**
5 **Assignment Operators**
6 **Bitwise Operators**
7 **Conditional Operator**
8 **Special Operators**

4.3.1 ARITHMETIC OPERATORS (+, -, *, /, %)

The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo. The following table provides information about arithmetic operators.

Operator	**Meaning**	**Example**
+	Addition	10 + 5 = 15
-	Subtraction	10 - 5 = 5
*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2
%	Remainder of the Division	5 % 2 = 1

⇒ The addition operator can be used with numerical data types and character data type. When it is used with numerical values, it performs mathematical addition and when it is used with character data type values, it performs concatination (appending).

⇒ The remainder of the division operator is used with integer data type only.

4.3.2 RELATIONAL OPERATORS (<, >, <=, >=, ==, !=)

The relational operators are the symbols that are used to compare two values. That means the relational operators are used to check the relationship between two values. Every relational operator has two results TRUE or FALSE. In simple words, the relational operators are used to define conditions in a program. The following table provides information about relational operators.

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE

4.3.3 LOGICAL OPERATORS (&&, ||, !)

The logical operators are the symbols that are used to combine multiple conditions into one condition. The following table provides information about logical operators.

Operator	Meaning	Example
&&	**Logical AND** - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
||	**Logical OR** - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 || 12 > 10 is TRUE
!	**Logical NOT** - Returns TRUE if condition is FLASE and returns FALSE if it is TRUE	!(10 < 5 && 12 > 10) is TRUE

⇒ **Logical AND -** Returns TRUE only if all conditions are TRUE, if any of the conditions is FALSE then complete condition becomes FALSE.
⇒ **Logical OR -** Returns FALSE only if all conditions are FALSE, if any of the conditions is TRUE then complete condition becomes TRUE.

4.3.4 INCREMENT & DECREMENT OPERATORS (++ & --)

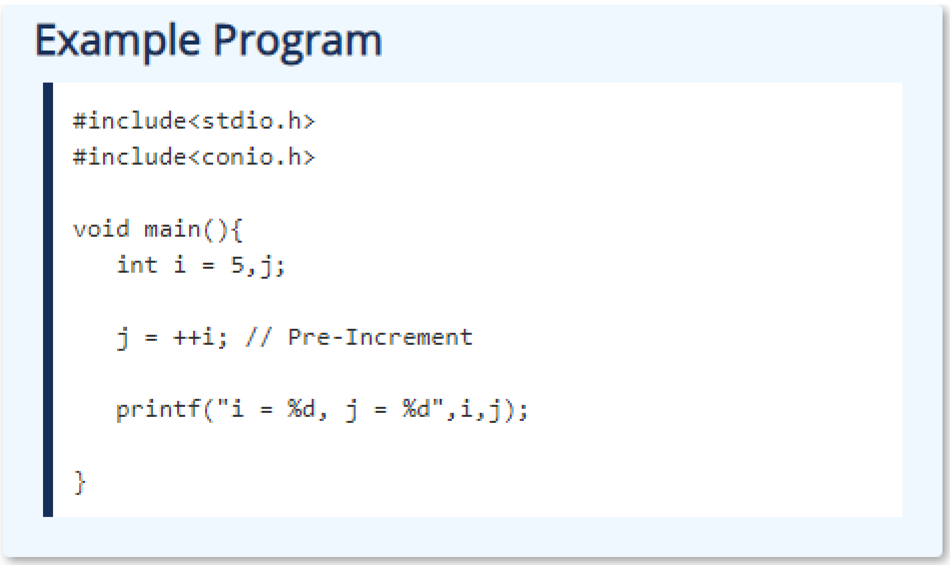
The increment and decrement operators are called unary operators because both need only one operand. The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand. The following table provides information about increment and decrement operators.

Operator	Meaning	Example
++	**Increment** - Adds one to existing value	int a = 5; a++; ⇒ a = 6
--	**Decrement** - Subtracts one from existing value	int a = 5; a--; ⇒ a = 4

The increment and decrement operators are used infront of the operand (++a) or after the operand (a++). If it is used infront of the operand, we call it as **pre-increment** or **pre-decrement** and if it is used after the operand, we call it as **post-increment** or **post-decrement**.

PRE-INCREMENT OR PRE-DECREMENT

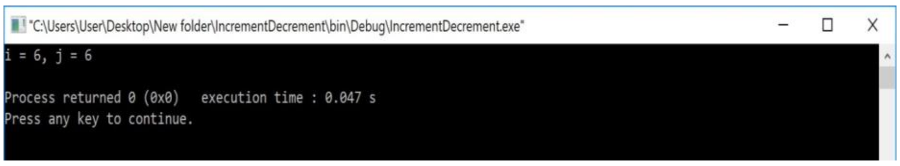
In the case of pre-increment, the value of the variable is increased by one before the expression evaluation. In the case of pre-decrement, the value of the variable is decreased by one before the expression evaluation. That means, when we use pre-increment or pre-decrement, first the value of the variable is incremented or decremented by one, then the modified value is used in the expression evaluation.



Example Program

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

void main(){
   int i = 5,j;

   j = ++i; // Pre-Increment

   printf("i = %d, j = %d",i,j);

}
```

Output:



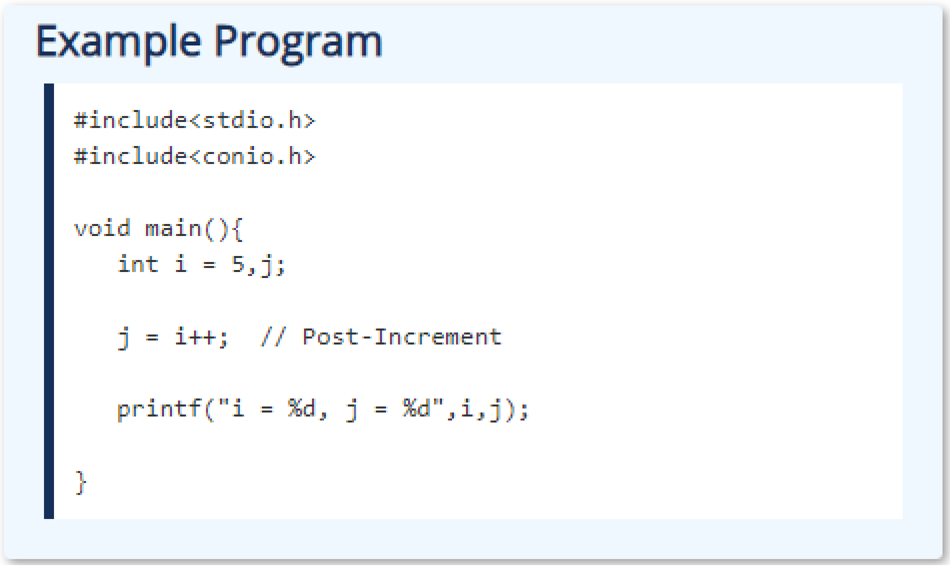
"C:\Users\User\Desktop\New folder\IncrementDecrement\bin\Debug\IncrementDecrement.exe"

```
i = 6, j = 6

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

POST-INCREMENT OR POST-DECREMENT

In the case of post-increment, the value of the variable is increased by one after the expression evaluation. In the case of post-decrement, the value of the variable is decreased by one after the expression evaluation.

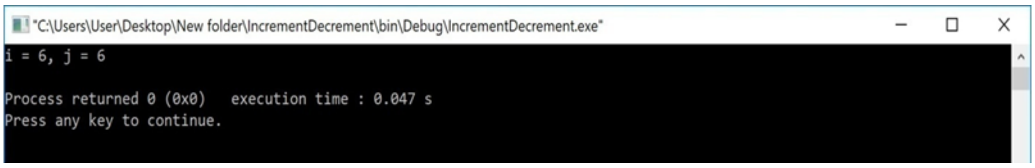
That means, when we use post-increment or post-decrement, first the expression is evaluated with existing value, then the value of the variable is incremented or decremented by one.



Example Program

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

void main(){
   int i = 5,j;

   j = i++;  // Post-Increment

   printf("i = %d, j = %d",i,j);

}
```

Output:



```
"C:\Users\User\Desktop\New folder\IncrementDecrement\bin\Debug\IncrementDecrement.exe"
i = 6, j = 6

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

4.3.5 ASSIGNMENT OPERATORS (=, +=, -=, *=, /=, %=)

The assignment operators are used to assign right-hand side value (Rvalue) to the left-hand side variable (Lvalue). The assignment operator is used in different variants along with arithmetic operators. The following table describes all the assignment operators in the C programming language.

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15
+=	Add both left and right-hand side values and store the result into left-hand side variable	A += 10 ⇒ A = A+10

-=	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	A -= B ⇒ A = A-B
***=**	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	A *= B ⇒ A = A*B
/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	A /= B ⇒ A = A/B
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	A %= B ⇒ A = A%B

4.3.6 BITWISE OPERATORS (&, |, ^, ~, >>, <<)

The bitwise operators are used to perform bit-level operations in the c programming language. When we use the bitwise operators, the operations are performed based on the binary values. The following table describes all the bitwise operators in the C programming language. Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100).

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
|	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A | B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A ⇒ 6 (00110)

<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 ⇒ 6 (00110)

4.3.7 CONDITIONAL OPERATOR (?:)

The conditional operator is also called a **ternary operator** because it requires three operands. This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result. If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed. The conditional operator is used with the following syntax.

Condition ? TRUE Part : FALSE Part;

Example

A = (10<15)?100:200; ⇒ A value is 100

Special Operators (sizeof, pointer, comma, dot, etc.)

The following are the special operators in **C** programming language.

4.3.8 SIZEOF OPERATOR

This operator is used to find the size of the memory (in bytes) allocated for a variable. This operator is used with the following syntax.

sizeof (variableName);

Example

sizeof (A); ⇒ the result is 2 if A is an integer

Pointer operator (*)

This operator is used to define pointer variables in C programming language.

Comma operator (,)

This operator is used to separate variables while they are declaring, separate the expressions in function calls, etc.

Dot operator (.)

This operator is used to access members of structure or union.

Check your Progress-2

3. Explain Arithmetic Operators

____

____

____

4. State Increment and decrement operator with example

____

____

____

4.4LET US SUM UP

We came across various Input functions and operators in C language programming and also understood clearly how to utilize these elements while writing a program.

4.5 KEYWORDS

Conditional operator - The conditional operator is also called a **ternary operator** because it requires three operands

Size of operator -This operator is used to find the size of the memory (in bytes) allocated for a variable.

Dot Operator - This operator is used to access members of structure or union.

4.6 QUESTIONS FOR REVIEW

1. Brief about input functions in C.

2. Explain Relational Operator

3. State different types of logical operators with meanings and examples.

4.7 SUGGESTED READINGS AND REFERENCES

1. Fundamentals of Real Analysis, S K. Berberian, Springer.
2. Measure Theory and Integration, G. De Barra, New Age International Publ.
3. Real Analysis, H. L. Royden.
4. Principles of Mathematical Analysis, W. Rudin.
5. Lectures on Real Analysis, J. Yeh, World Sci.
6. R. G. Bartle, The Elements of Integration, John Wiley & Sons, Inc. New York, 1966

4.8ANSWERS TO CHECK YOUR PROGRESS

1. Provide explanation, syntax and example- 4.2.2

2. Provide definition – 4.2.5

3.Provide explanation with example – 4.3.1

4. Provide table stating operator, meaning and example– 4.3.4

UNIT-5 EXPRESSION AND ‘IF’ STATEMENT IN C

STRUCTURE

5.0 OBJECTIVES

Understand the Expression and Expression Types in C

Understand the concept of C Operator Precedence and Associativity

Enumerate Type Casting and Conversion In C

Comprehend the application of 'IF' statement in C

5.1 INTRODUCTION

In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression.

5.2 C EXPRESSIONS

What is an expression?

In the C programming language, an expression is defined as follows.
An expression is a collection of operators and operands that represents a specific value. In the above definition, an **operator** is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.
Operands are the values on which the operators perform the task. Here operand can be a direct value or variable or address of memory location.

5.3 EXPRESSION TYPES IN C

In the C programming language, expressions are divided into THREE types. They are as follows...

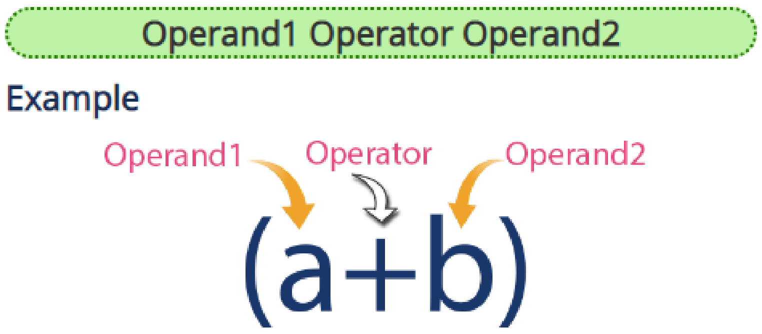
1. **Infix Expression**
2. **Postfix Expression**
3. **Prefix Expression**

The above classification is based on the operator position in the expression.

5.3.1 INFIX EXPRESSION

The expression in which the operator is used between operands is called infix expression.

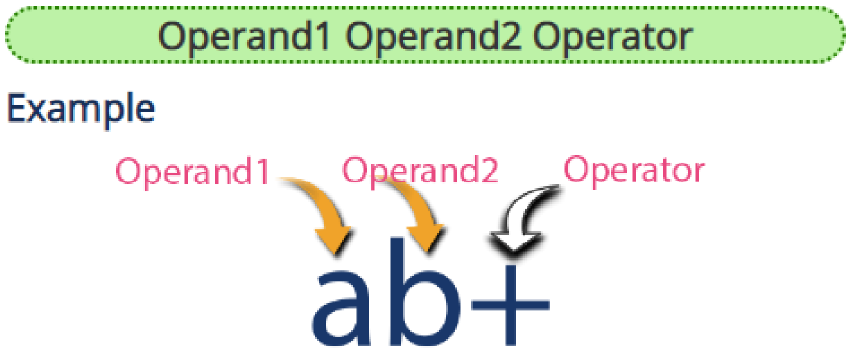
The infix expression has the following general structure.



5.3.2 Postfix Expression

The expression in which the operator is used after operands is called postfix expression.

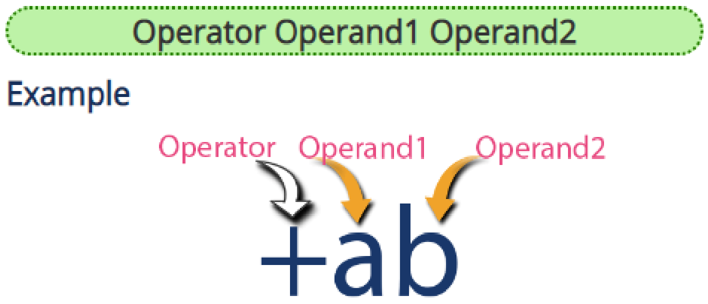
The postfix expression has the following general structure.



5.3.3 PREFIX EXPRESSION

The expression in which the operator is used before operands is called a prefix expression.

The prefix expression has the following general structure.



Check your Progress-1

1. Define Expression In C

____

____

____

2. Explain postfix Expession

____

____

____

5.4 C OPERATOR PRECEDENCE AND ASSOCIATIVITY

What is Operator Precedence?

Operator precedence is used to determine the order of operators evaluated in an expression. In C programming language every operator has precedence (priority). When there is more than one operator in an expression the operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.

What is Operator Associativity?

Operator associativity is used to determine the order of operators with equal precedence evaluated in an expression. In the c programming language, when an expression contains multiple operators with equal precedence, we use associativity to determine the order of evaluation of those operators.

In C programming language the operator precedence and associativity are as shown in the following table.

Precedence	Operator	Operator Meaning	Associativity
1	() []	function call array reference	Left to Right

	-> .	structure member access structure member access	
2	! ~ + - ++ -- & * sizeof (type)	negation 1's complement Unary plus Unary minus increment operator decrement operator address of operator pointer returns size of a variable type conversion	Right to Left
3	* / %	multiplication division remainder	Left to Right
4	+ -	addition subtraction	Left to Right
5	<< >>	left shift right shift	Left to Right
6	< <= > >=	less than less than or equal to greater than greater than or equal to	Left to Right
7	== !=	equal to not equal to	Left to Right
8	&	bitwise AND	Left to Right
9	^	bitwise EXCLUSIVE OR	Left to Right
10	|	bitwise OR	Left to Right
11	&&	logical AND	Left to Right
12	||	logical OR	Left to Right
13	?:	conditional operator	Left to Right
14	= *= /= %=	assignment assign multiplication assign division assign remainder	Right to Left

	+=	assign addition	
	-=	assign subtraction	
	&=	assign bitwise AND	
	^=	assign bitwise XOR	
	|=	assign bitwise OR	
	<<=	assign left shift	
	>>=	assign right shift	
15	,	separator	Left to Right

In the above table, the operator precedence decreases from top to bottom and increases from bottom to top.

5.5 C EXPRESSION EVALUATION

In the C programming language, an expression is evaluated based on the operator precedence and associativity. When there are multiple operators in an expression, they are evaluated according to their precedence and associativity. The operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.
An expression is evaluated based on the precedence and associativity of the operators in that expression.

To understand expression evaluation in c, let us consider the following simple example expression...

```
10 + 4 * 3 / 2
```

In the above expression, there are three operators **+, * and** /. Among these three operators, both multiplication and division have the same higher precedence and addition has lower precedence. So, according to the operator precedence both multiplication and division are evaluated first and then the addition is evaluated. As multiplication and division have the same precedence they are evaluated based on the associativity. Here, the associativity of multiplication and division is **left to right**. So, multiplication is performed first, then division and finally addition. So, the above expression is evaluated in the order of *** / and +**. It is evaluated as follows...

4 * 3 ====> 12

12 / 2 ===> 6

10 + 6 ===> 16

The expression is evaluated to **16**.

5.6 TYPE CASTING AND CONVERSION IN C

In a programming language, the expression contains data values of the same datatype or different data types. When the expression contains similar datatype values then it is evaluated without any problem. But if the expression contains two or more different datatype values then they must be converted to the single datatype of destination datatype. Here, the destination is the location where the final result of that expression is stored. For example, the multiplication of an integer data value with the float data value and storing the result into a float variable. In this case, the integer value must be converted to float value so that the final result is a float datatype value. In a **C** programming language, the data conversion is performed in two different methods as follows...

1. Type Conversion
2. Type Casting

5.6.1 TYPE CONVERSION

The type conversion is the process of converting a data value from one data type to another data type automatically by the compiler. Sometimes type conversion is also called **implicit type conversion**. The implicit type conversion is automatically performed by the compiler. For example, in C programming language, when we assign an integer value to a float variable the integer value automatically gets converted to float value by adding decimal value 0. And when a float value is assigned to an integer variable the float value automatically gets converted to an integer value by removing the decimal value. To understand more about type conversion observe the following...

int i = 10 ;

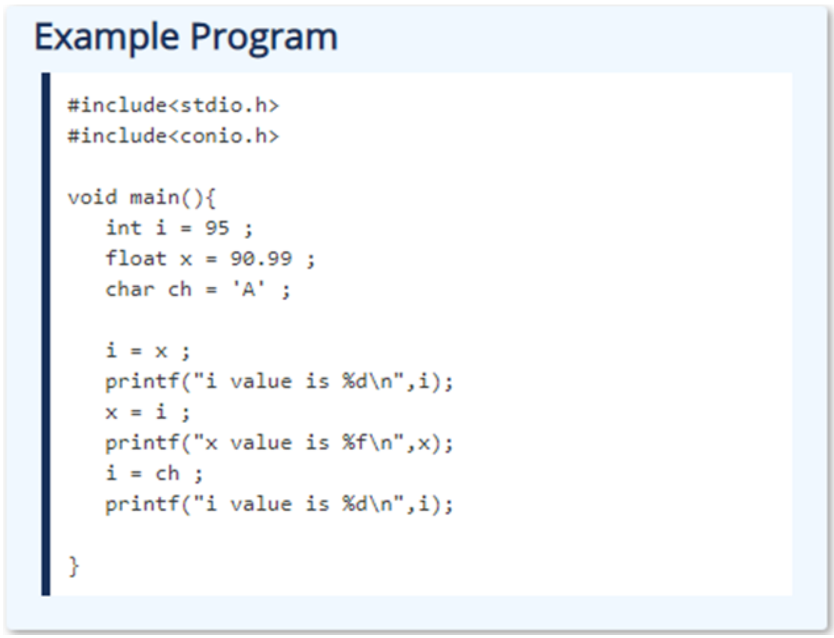
float x = 15.5 ;

char ch = 'A' ;

i = x ; =======> x value 15.5 is converted as 15 and assigned to variable i

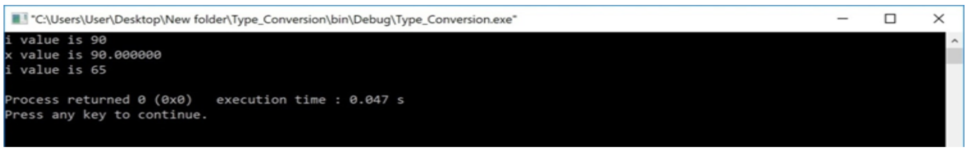
x = i ; =======> Here i value 10 is converted as 10.000000 and assigned to variable x

i = ch ; =======> Here the ASCII value of A (65) is assigned to i



Example Program

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

void main(){
   int i = 95 ;
   float x = 90.99 ;
   char ch = 'A' ;

   i = x ;
   printf("i value is %d\n",i);
   x = i ;
   printf("x value is %f\n",x);
   i = ch ;
   printf("i value is %d\n",i);

}
```



"C:\Users\User\Desktop\New folder\Type_Conversion\bin\Debug\Type_Conversion.exe"

```
i value is 90
x value is 90.000000
i value is 65

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

In the above program, we assign **i = x**, i.e., float variable value is assigned to the integer variable. Here, the compiler automatically converts the float value (90.99) into integer value (90) by removing the decimal part of the float value (90~~.99~~) and then it is assigned to variable **i**. Similarly, when we assign **x = i**, the integer value (90) gets converted to float value (90.000000) by adding zero as the decimal part.

5.6.2 TYPECASTING

Typecasting is also called an **explicit type conversion**. Compiler converts data from one data type to another data type implicitly. When compiler converts implicitly, there may be a data loss. In such a case, we convert the data from one data type to another data type using explicit type conversion. To perform this we use the **unary cast operator**. To convert data from one type to another type we specify the target data type in parenthesis as a prefix to the data value that has to be converted. The general syntax of typecasting is as follows.

(TargetDatatype) DataValue

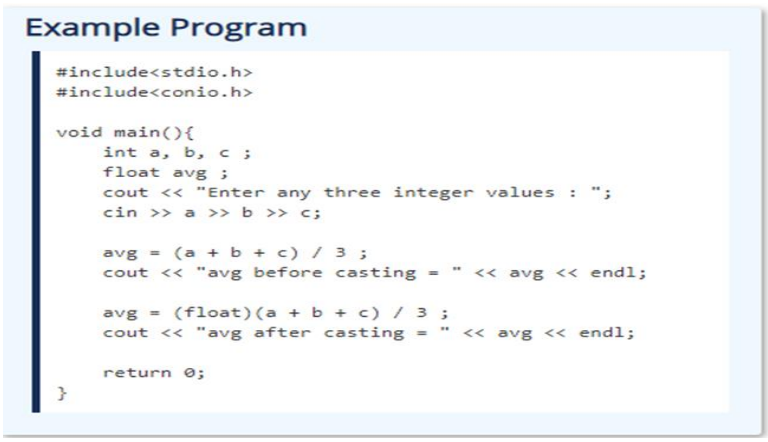
Example

int totalMarks = 450, maxMarks = 600 ;

float average ;

average = (float) totalMarks / maxMarks * 100 ;

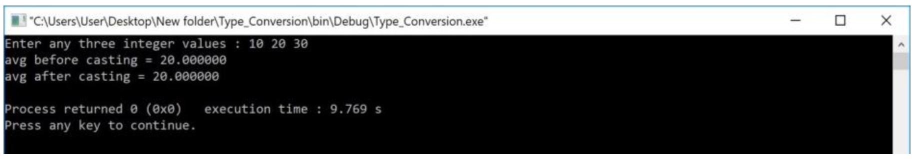
In the above example code, both totalMarks and maxMarks are integer data values. When we perform totalMarks / maxMarks the result is a float value, but the destination (average) datatype is a float. So we use type casting to convert totalMarks and maxMarks into float data type.



Example Program

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

void main(){
    int a, b, c ;
    float avg ;
    cout << "Enter any three integer values : ";
    cin >> a >> b >> c;

    avg = (a + b + c) / 3 ;
    cout << "avg before casting = " << avg << endl;

    avg = (float)(a + b + c) / 3 ;
    cout << "avg after casting = " << avg << endl;

    return 0;
}
```

Output:



"C:\Users\User\Desktop\New folder\Type_Conversion\bin\Debug\Type_Conversion.exe"

```
Enter any three integer values : 10 20 30
avg before casting = 20.000000
avg after casting = 20.000000

Process returned 0 (0x0)   execution time : 9.769 s
Press any key to continue.
```

Check your Progress-2

3. What is Operator Precedence

____

____

____

4. Discuss - an expression is evaluated based on the operator precedence and associativity.

____

____

____

5.7 'IF' STATEMENT IN C

In the C programming language, the program execution flow is line by line from top to bottom. That means the C program is executed line by line from the main method. But this type of execution flow may not be suitable for all the program solutions. Sometimes, we make some decisions or we may skip the execution of one or more lines of code. Consider a situation, where we write a program to check whether a student has passed or failed in a particular subject. Here, we need to check whether the marks are greater than the pass marks or not. If marks are greater, then we decide that the student has passed otherwise failed. To solve such kind of problems in C we use the statements called decision making statements.

Decision-making statements are the statements that are used to verify a given condition and decide whether a block of statements gets executed or not based on the condition result.

In the C programming language, there are two decision-making statements they are as follows.

1. **if statement**
2. **switch statement**

if statement in C

In c, if statement is used to make decisions based on a condition. The if statement verifies the given condition and decides whether a block of statements are executed or not based on the condition result. In c, if statement is classified into four types as follows...

1. **Simple if statement**
2. **if-else statement**
3. **Nested if statement**
4. **if-else-if statement (if-else ladder)**

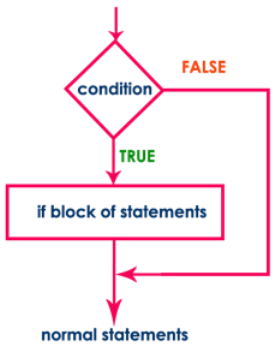
5.7.1 Simple if statement

Simple if statement is used to verify the given condition and executes the block of statements based on the condition result. The simple if statement evaluates specified condition. If it is TRUE, it executes the next statement or block of statements. If the condition is FALSE, it skips the execution of the next statement or block of statements. The general syntax and execution flow of the simple if statement is as follows.

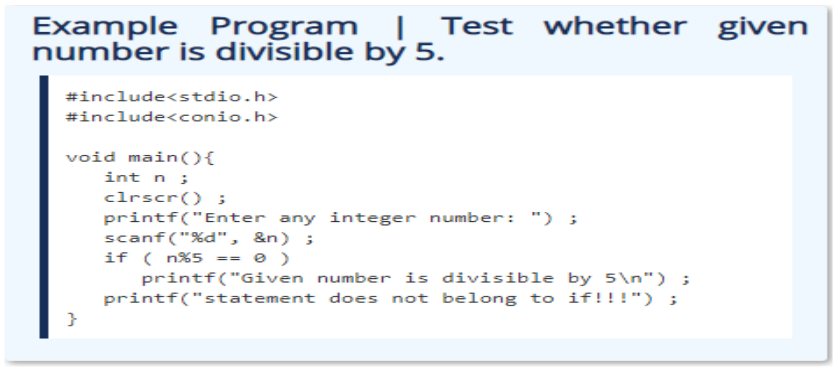
Syntax

```
if ( condition )
{
   ....
   block of statements;
   ....
}
```

Execution flow diagram



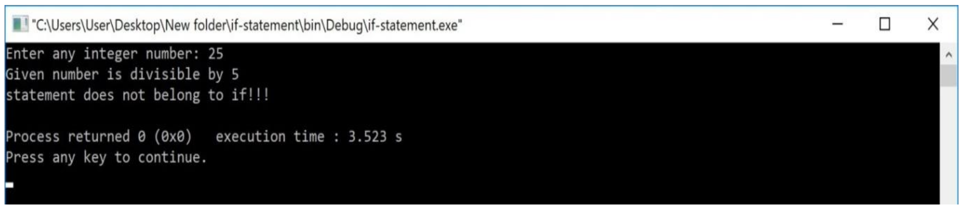
Simple if statement is used when we have only one option that is executed or skipped based on a condition.



Example Program | Test whether given number is divisible by 5.

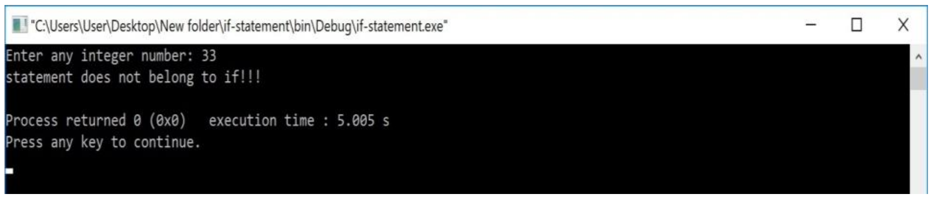
```
#include<stdio.h>
#include<conio.h>

void main(){
   int n ;
   clrscr() ;
   printf("Enter any integer number: ") ;
   scanf("%d", &n) ;
   if ( n%5 == 0 )
      printf("Given number is divisible by 5\n") ;
   printf("statement does not belong to if!!!") ;
}
```

Output 1:



```
"C:\Users\User\Desktop\New folder\if-statement\bin\Debug\if-statement.exe"
Enter any integer number: 25
Given number is divisible by 5
statement does not belong to if!!!

Process returned 0 (0x0)   execution time : 3.523 s
Press any key to continue.
```

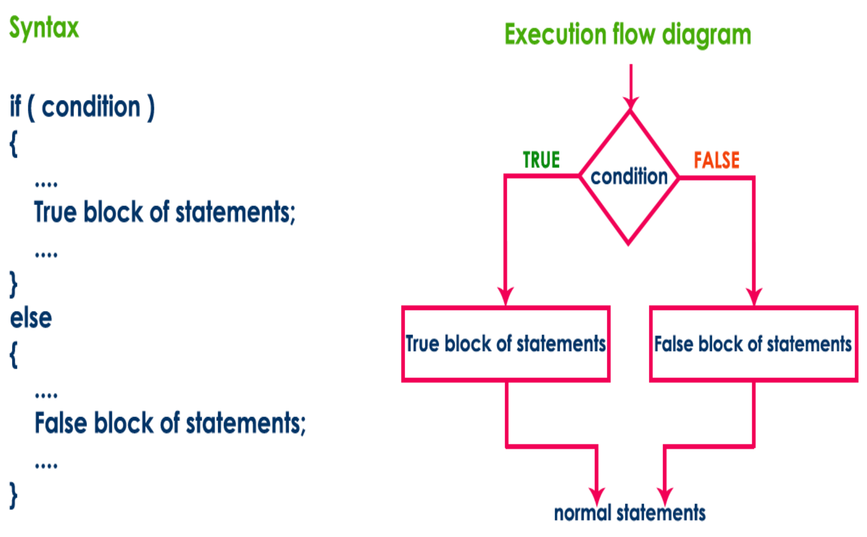
Output 2:



```
"C:\Users\User\Desktop\New folder\if-statement\bin\Debug\if-statement.exe"
Enter any integer number: 33
statement does not belong to if!!!

Process returned 0 (0x0)   execution time : 5.005 s
Press any key to continue.
```

5.7.2 if-else statement

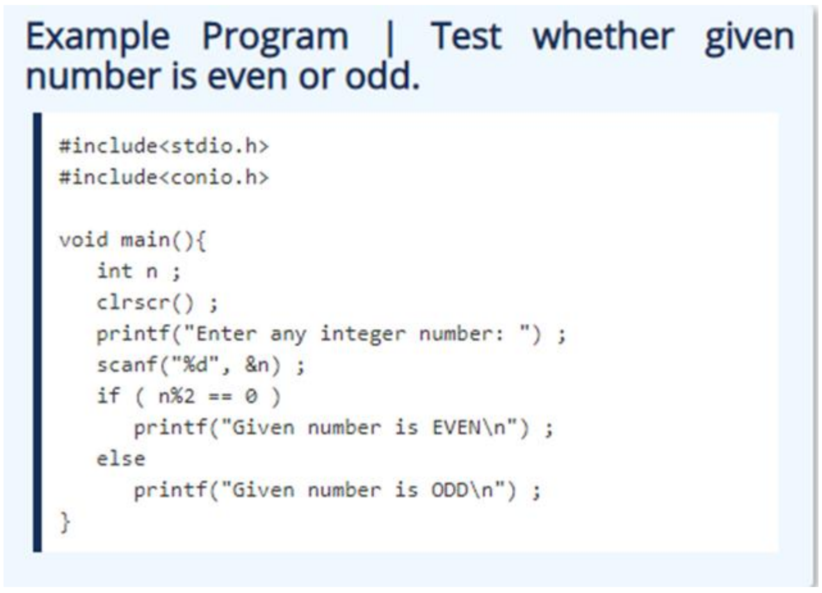
The if-else statement is used to verify the given condition and executes only one out of the two blocks of statements based on the condition result. The if-else statement evaluates the specified condition. If it is TRUE, it executes a block of statements (True block). If the condition is FALSE, it executes another block of statements (False block). The general syntax and execution flow of the if-else statement is as follows.



Syntax

```
if ( condition )
{
    ....
    True block of statements;
    ....
}
else
{
    ....
    False block of statements;
    ....
}
```

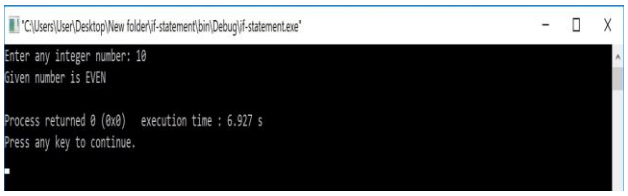
The if-else statement is used when we have two options and only one option has to be executed based on a condition result (TRUE or FALSE).



Example Program | Test whether given number is even or odd.

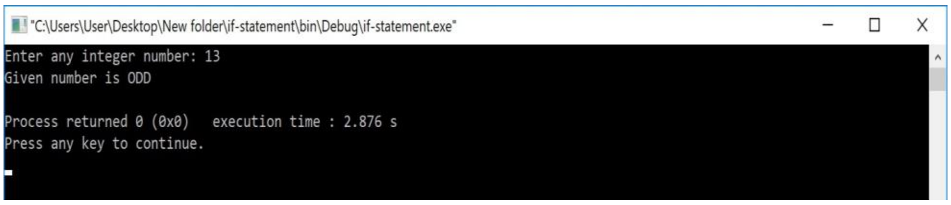
```
#include<stdio.h>
#include<conio.h>

void main(){
   int n ;
   clrscr() ;
   printf("Enter any integer number: ") ;
   scanf("%d", &n) ;
   if ( n%2 == 0 )
      printf("Given number is EVEN\n") ;
   else
      printf("Given number is ODD\n") ;
}
```

Output 1:



```
"C:\Users\User\Desktop\New folder\if-statement\bin\Debug\if-statement.exe"
Enter any integer number: 10
Given number is EVEN

Process returned 0 (0x0)   execution time : 6.927 s
Press any key to continue.
```

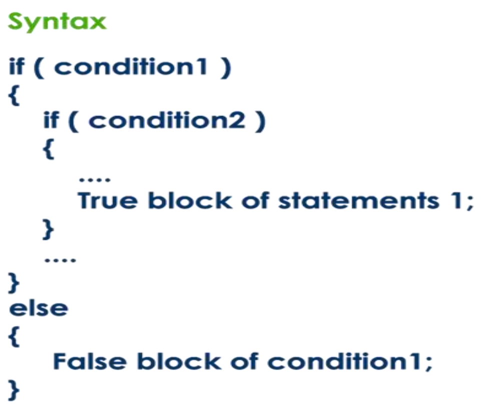
Output 2:



```
"C:\Users\User\Desktop\New folder\if-statement\bin\Debug\if-statement.exe"
Enter any integer number: 13
Given number is ODD

Process returned 0 (0x0)   execution time : 2.876 s
Press any key to continue.
```

5.7.3 Nested if statement

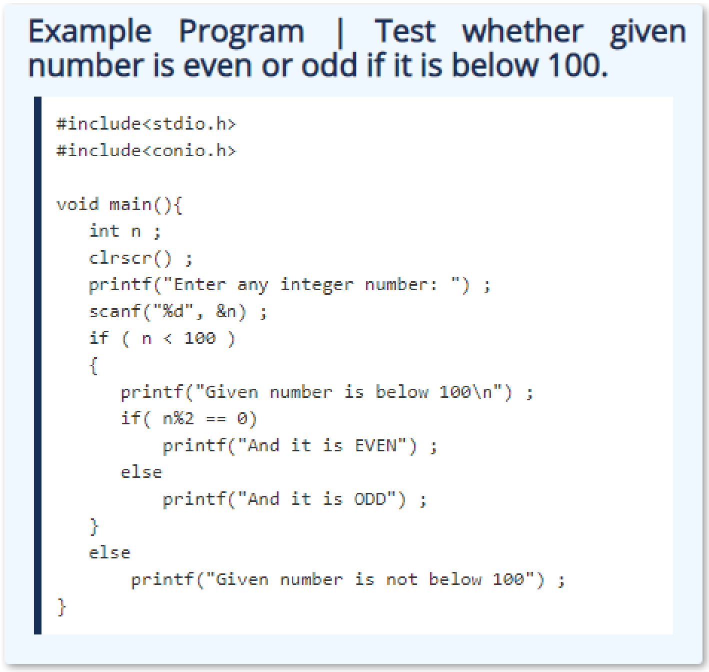
Writing a if statement inside another if statement is called nested if statement. The general syntax of the nested if statement is as follows...



Syntax

```
if ( condition1 )
{
   if ( condition2 )
   {
      ....
      True block of statements 1;
   }
   ....
}
else
{
    False block of condition1;
}
```

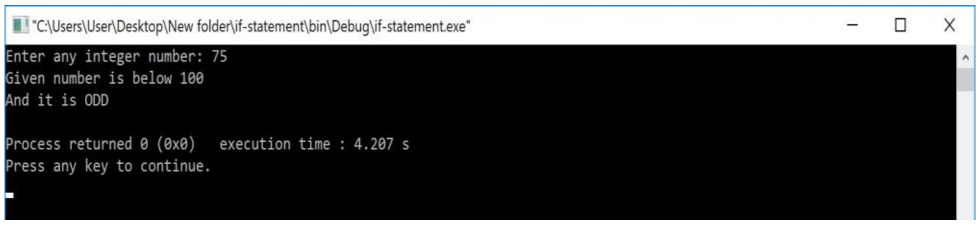
The nested if statement can be defined using any combination of simple if & if-else statements.



Example Program | Test whether given number is even or odd if it is below 100.

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

void main(){
   int n ;
   clrscr() ;
   printf("Enter any integer number: ") ;
   scanf("%d", &n) ;
   if ( n < 100 )
   {
      printf("Given number is below 100\n") ;
      if( n%2 == 0)
          printf("And it is EVEN") ;
      else
          printf("And it is ODD") ;
   }
   else
       printf("Given number is not below 100") ;
}
```

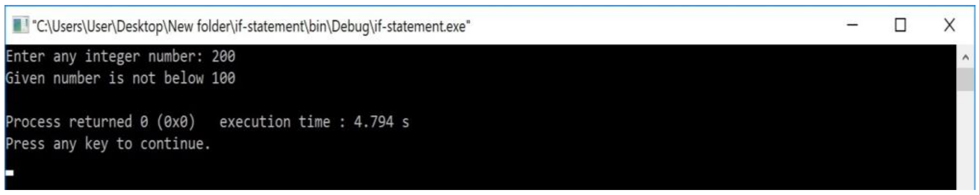
Output 1:



"C:\Users\User\Desktop\New folder\if-statement\bin\Debug\if-statement.exe"

```
Enter any integer number: 75
Given number is below 100
And it is ODD

Process returned 0 (0x0)   execution time : 4.207 s
Press any key to continue.
```

Output 2:



"C:\Users\User\Desktop\New folder\if-statement\bin\Debug\if-statement.exe"

```
Enter any integer number: 200
Given number is not below 100

Process returned 0 (0x0)   execution time : 4.794 s
Press any key to continue.
```

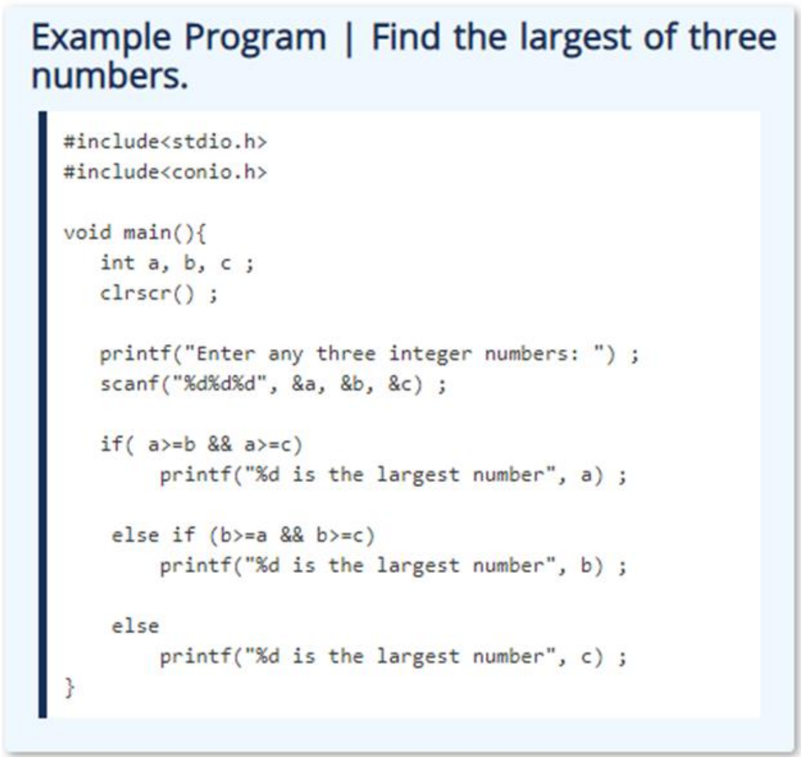
5.7.4 if-else-if statement (if-else ladder)

Writing a if statement inside else of an if statement is called if-else-if statement. The general syntax of the if-else-if statement is as follows...

Syntax

```
if ( condition1 )
{
    ....
    True block of statements1;
    ....
}
else if ( condition2 )
{
    False block of condition1;
    &
    True block of condition2
}
```

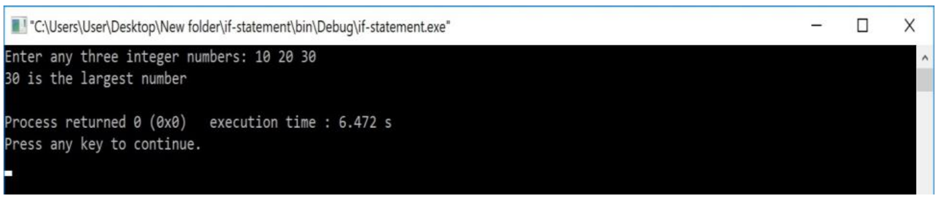
The if-else-if statement can be defined using any combination of simple if & if-else statements.



Example Program | Find the largest of three numbers.

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

void main(){
   int a, b, c ;
   clrscr() ;

   printf("Enter any three integer numbers: ") ;
   scanf("%d%d%d", &a, &b, &c) ;

   if( a>=b && a>=c)
        printf("%d is the largest number", a) ;

    else if (b>=a && b>=c)
        printf("%d is the largest number", b) ;

    else
        printf("%d is the largest number", c) ;
}
```

Output:



"C:\Users\User\Desktop\New folder\if-statement\bin\Debug\if-statement.exe"

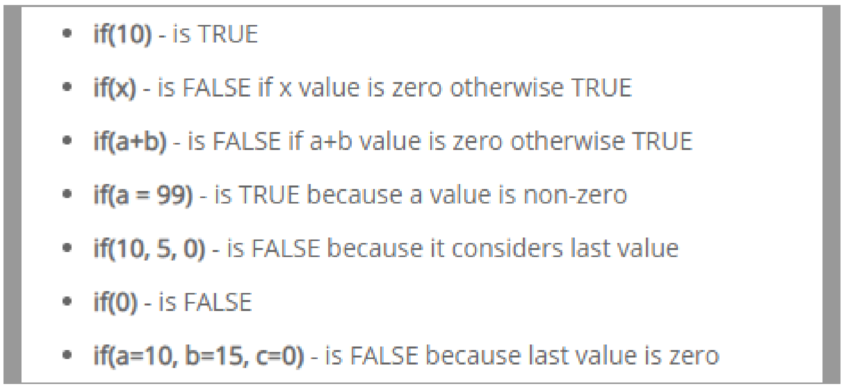
```
Enter any three integer numbers: 10 20 30
30 is the largest number

Process returned 0 (0x0)   execution time : 6.472 s
Press any key to continue.
```

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use a conditional control statement like if statement, the condition might be an expression evaluated to a numerical value, a variable or a direct numerical value.

If the expression value or direct value is zero the condition becomes FALSE otherwise becomes TRUE.

To understand more consider the following statements.



- **if(10)** - is TRUE
- **if(x)** - is FALSE if x value is zero otherwise TRUE
- **if(a+b)** - is FALSE if a+b value is zero otherwise TRUE
- **if(a = 99)** - is TRUE because a value is non-zero
- **if(10, 5, 0)** - is FALSE because it considers last value
- **if(0)** - is FALSE
- **if(a=10, b=15, c=0)** - is FALSE because last value is zero

5.8 LET US SUM UP

If statements are decision making statements that helps to solve number of problems. If statements provides more applicability to C programming in real world problems

5.9 KEYWORDS

Task - a piece of work to be done or undertaken.

Evaluated - find a numerical expression or equivalent for (an equation, formula, or function).

Type conversion- is the process of converting a data value from one data type to another data type automatically by the compiler

5.10 QUESTIONS FOR REVIEW

1. Explain types of expression in C.

2. Explain Operator Precedence & Associativity with the help of operators and examples.

3. How the data conversion takes place in C ?

4. Enumerate 'If' statement in C.

5.11 SUGGESTED READINGS AND REFERENCES

1. Fundamentals of Real Analysis, S K. Berberian, Springer.
2. Measure Theory and Integration, G. De Barra, New Age International Publ.
3. Real Analysis, H. L. Royden.
4. Principles of Mathematical Analysis, W. Rudin.
5. Lectures on Real Analysis, J. Yeh, World Sci.
6. R. G. Bartle, The Elements of Integration, John Wiley & Sons, Inc. New York, 1966

5.12 ANSWERS TO CHECK YOUR PROGRESS

1. Provide explanation, syntax and example- 5.2

2. Provide explanation, syntax and example – 5.3.2

3.Provide explanation – 5.4

4. Provide explanation – 5.5

UNIT-6 DIFFERENT STATEMENT IN C

STRUCTURE

6.0 OBJECTIVES

Understand the Switch, while and do- while statement in C

Understand the for, break, continue and goto in C

6.1 INTRODUCTION

C provides two styles of flow control:

- Branching
- Looping

Branching is deciding what actions to take and looping is deciding how many times to take a certain action

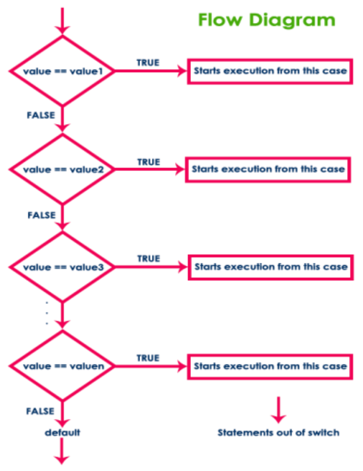
6.2 'SWITCH' STATEMENT IN C

Consider a situation in which we have many options out of which we need to select only one option that is to be executed. Such kind of problems can be solved using **nested if** statement. But as the number of options increases, the complexity of the program also gets increased. This type of problem can be solved very easily using a **switch** statement. Using the switch statement, one can select only one option from more number of options very easily. In the switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches the value associated with an option, the execution starts from that option. In the switch statement, every option is defined as a **case**.

The switch statement has the following syntax and execution flow diagram.

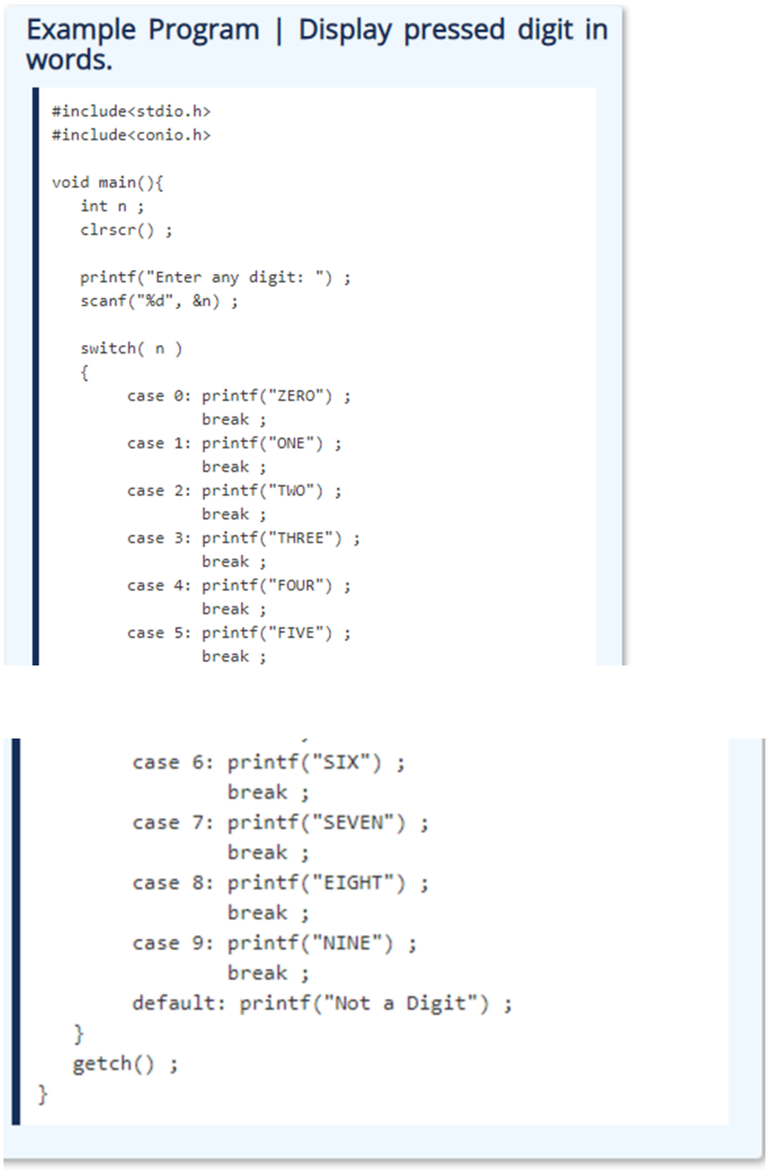
Syntax

```
switch ( expression or value )
{
      case  value1:  set of statements;
                     ....
      case  value2:  set of statements;
                     ....
      case  value3:  set of statements;
                     ....
      case  value4:  set of statements;
                     ....
      case  value5:  set of statements;
                     ....
      .
      .
      default: set of statements;
}
```



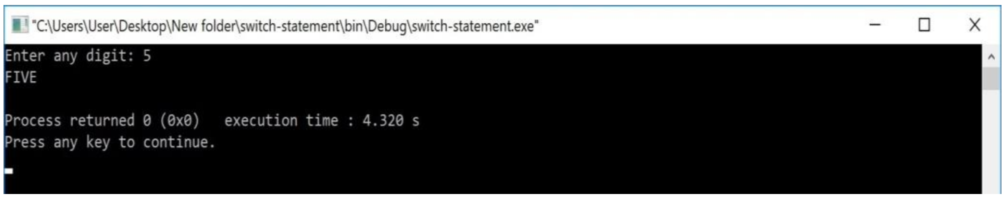
The switch statement contains one or more cases and each case has a value associated with it. At first switch statement compares the first case value with the switchValue, if it gets matched the execution starts from the first case. If it doesn't match the switch statement compares the second case value with the switchValue and if it is matched the execution starts from the second case. This process continues until it finds a match. If no case value matches with the switchValue specified in the switch statement, then a special case called **default** is executed.

When a case value matches with the switchValue, the execution starts from that particular case. This execution flow continues with the next case statements also. To avoid this, we use the "**break**" statement at the end of each case. That means the **break** statement is used to terminate the switch statement. However, it is optional.


Example Program | Display pressed digit in words.

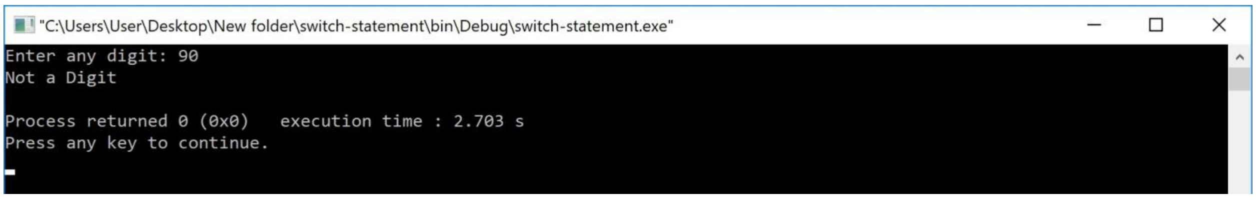
```
#include<stdio.h>
#include<conio.h>

void main(){
   int n ;
   clrscr() ;

   printf("Enter any digit: ") ;
   scanf("%d", &n) ;

   switch( n )
   {
        case 0: printf("ZERO") ;
                break ;
        case 1: printf("ONE") ;
                break ;
        case 2: printf("TWO") ;
                break ;
        case 3: printf("THREE") ;
                break ;
        case 4: printf("FOUR") ;
                break ;
        case 5: printf("FIVE") ;
                break ;
        case 6: printf("SIX") ;
                break ;
        case 7: printf("SEVEN") ;
                break ;
        case 8: printf("EIGHT") ;
                break ;
        case 9: printf("NINE") ;
                break ;
        default: printf("Not a Digit") ;
   }
   getch() ;
}
```

Output 1:



```
"C:\Users\User\Desktop\New folder\switch-statement\bin\Debug\switch-statement.exe"
Enter any digit: 5
FIVE

Process returned 0 (0x0)   execution time : 4.320 s
Press any key to continue.
```

Output 2:



```
"C:\Users\User\Desktop\New folder\switch-statement\bin\Debug\switch-statement.exe"
Enter any digit: 90
Not a Digit

Process returned 0 (0x0)   execution time : 2.703 s
Press any key to continue.
```

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use switch statement, we must follow the following...

- Both **switch** and **case** are keywords so they must be used only in lower case letters.
- The data type of case value and the value specified in the switch statement must be the same.
- switch and case values must be either integer or character but not float or string.
- A switch statement can contain any number of cases.
- The keyword **case** and its value must be superated with a white space.
- The case values need not be defined in sequence, they can be in any order.
- The **default** case is optional and it can be defined anywhere inside the switch statement.
- The switch value might be direct, a variable or an expression.

6.3 'WHILE' STATEMENT IN C

Consider a situation in which we execute a single statement or block of statements repeatedly for the required number of times. Such kind of problems can be solved using **looping** statements in C. For example, assume a situation where we print a message 100 times. If we want to perform that task without using looping statements, we have to either write 100 printf statements or we have to write the same message 100 times in a single printf statement. Both are complex methods. The same task can be performed very easily using looping statements.

The looping statements are used to execute a single statement or block of statements repeatedly until the given condition is FALSE.

C language provides three looping statements...

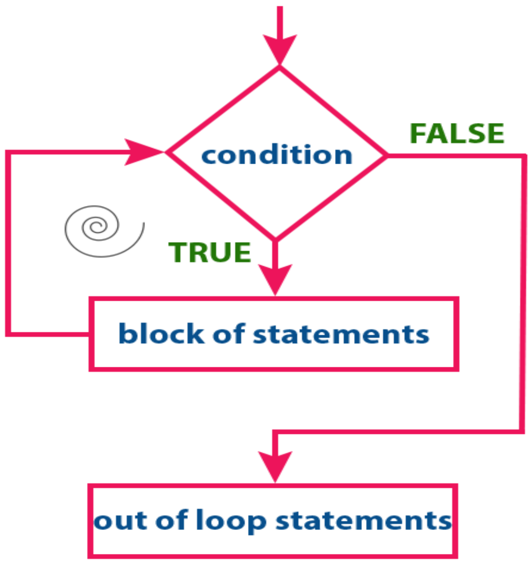
- **while statement**
- **do-while statement**
- **for statement**

while Statement

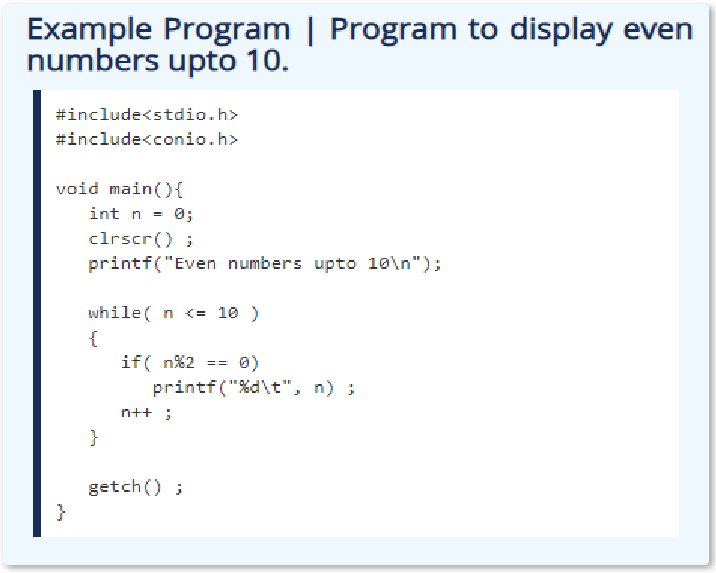
The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as **Entry control looping statement**. The while statement has the following syntax...

Syntax:

```
while( condition )
{
        ...
        block of statements;
        ...
}
```



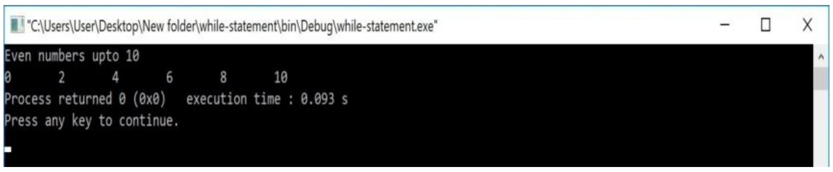
At first, the given condition is evaluated. If the condition is TRUE, the single statement or block of statements gets executed. Once the execution gets completed the condition is evaluated again. If it is TRUE, again the same statements get executed. The same process is repeated until the condition is evaluated to FALSE. Whenever the condition is evaluated to FALSE, the execution control moves out of the while block.



Example Program | Program to display even numbers upto 10.

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

void main(){
   int n = 0;
   clrscr() ;
   printf("Even numbers upto 10\n");

   while( n <= 10 )
   {
      if( n%2 == 0)
         printf("%d\t", n) ;
      n++ ;
   }

   getch() ;
}
```

Output:



"C:\Users\User\Desktop\New folder\while-statement\bin\Debug\while-statement.exe"

```
Even numbers upto 10
0       2       4       6       8       10
Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

MOST IMPORTANT POINTS TO BE REMEMBERED

when we use a while statement, we must follow the following...

- **while** is a keyword so it must be used only in lower case letters.
- If the condition contains a variable, it must be assigned a value before it is used.
- The value of the variable used in condition must be modified according to the requirement inside the while block.

- In a while statement, the condition may be a direct integer value, a variable or a condition.
- A while statement can be an empty statement.

Check your Progress-1

1.What are the important points need to be remembered while using switch statement

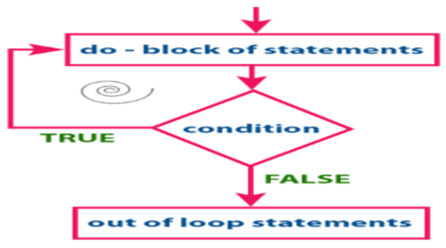
2. Define looping statements in C

6.4 'DO-WHILE' STATEMENT IN C

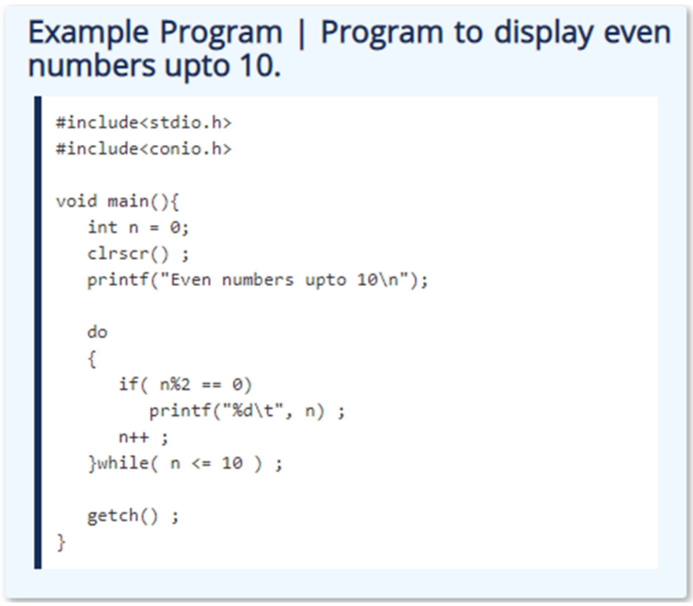
The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE. The do-while statement is also known as the **Exit control looping statement**. The do-while statement has the following syntax...

```
Syntax:

do
{
      ...
      block of statements;
      ...
} while( condition ) ;
```

The do-while statement has the following execution flow diagram...

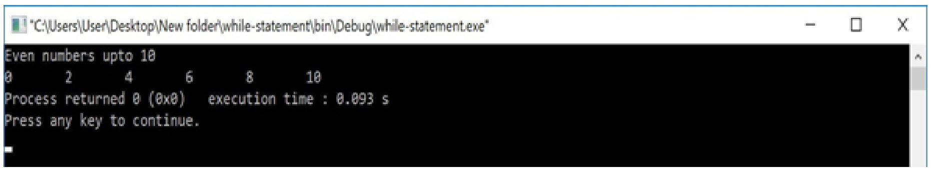


At first, the single statement or block of statements which are defined in **do** block are executed. After the execution of the do block, the given condition gets evaluated. If the condition is evaluated to TRUE, the single statement or block of statements of do block are executed again. Once the execution gets completed again the condition is evaluated. If it is TRUE, again the same statements are executed. The same process is repeated until the condition is evaluated to FALSE. Whenever the condition is evaluated to FALSE, the execution control moves out of the while block.


Example Program | Program to display even numbers upto 10.

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

void main(){
   int n = 0;
   clrscr() ;
   printf("Even numbers upto 10\n");

   do
   {
      if( n%2 == 0)
         printf("%d\t", n) ;
      n++ ;
   }while( n <= 10 ) ;

   getch() ;
}
```

Output:


"C:\Users\User\Desktop\New folder\while-statement\bin\Debug\while-statement.exe"

```
Even numbers upto 10
0       2       4       6       8       10
Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use the do-while statement, we must follow the following...

- Both **do** and **while** are keywords so they must be used only in lower case letters.
- If the condition contains a variable, it must be assigned a value before it is used.
- The value of the variable used in the condition must be modified according to the requirement inside the do block.

- In a do-while statement, the condition may be a direct integer value, a variable or a condition.
- A do-while statement can be an empty statement.
- **In do-while, the block of statements is executed at least once**.

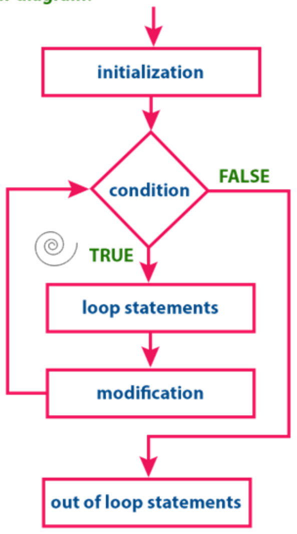
6.5 'FOR' STATEMENT IN C

The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE. The for statement has the following syntax and execution flow diagram...

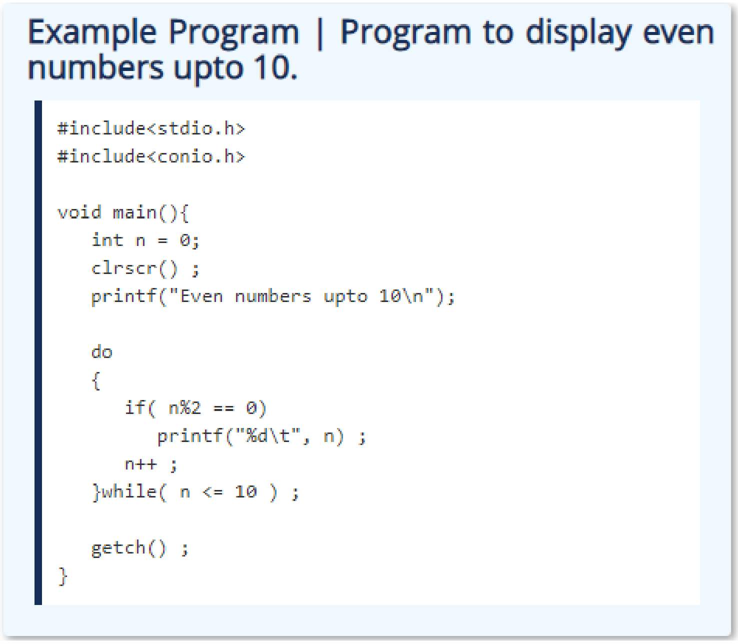
Syntax:

```
for( initialization ; condition ; modification )
{
    ...
    block of statements;
    ...
}
```

Execution flow diagram:



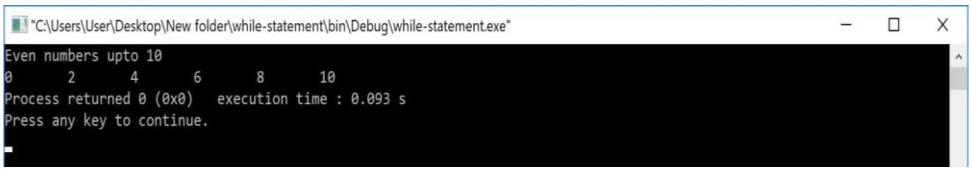
At first, the for statement executes **initialization** followed by **condition** evaluation. If the condition is evaluated to TRUE, the single statement or block of statements of for statement are executed. Once the execution gets completed, the **modification** statement is executed and again the condition is evaluated. If it is TRUE, again the same statements are executed. The same process is repeated until the condition is evaluated to FALSE. Whenever the condition is evaluated to FALSE, the execution control moves out of the for block.



Example Program | Program to display even numbers upto 10.

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

void main(){
   int n = 0;
   clrscr() ;
   printf("Even numbers upto 10\n");

   do
   {
      if( n%2 == 0)
         printf("%d\t", n) ;
      n++ ;
   }while( n <= 10 ) ;

   getch() ;
}
```

Output:



```
"C:\Users\User\Desktop\New folder\while-statement\bin\Debug\while-statement.exe"
Even numbers upto 10
0       2       4       6       8       10
Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use for statement, we must follow the following...

- **for** is a keyword so it must be used only in lower case letters.
- Every for statement must be provided with initialization, condition, and modification (They can be empty but must be separated with ";")
  Ex: **for (; ;)** or **for (; condition ; modification)** or **for (; condition ;)**
- In for statement, the condition may be a direct integer value, a variable or a condition.
- The for statement can be an empty statement.

6.6 BREAK, CONTINUE AND GOTO IN C

In **C**, there are control statements that do not need any condition to control the program execution flow. These control statements are called as **unconditional control statements**. **C** programming language provides the following unconditional control statements...

- **break**
- **continue**
- **goto**

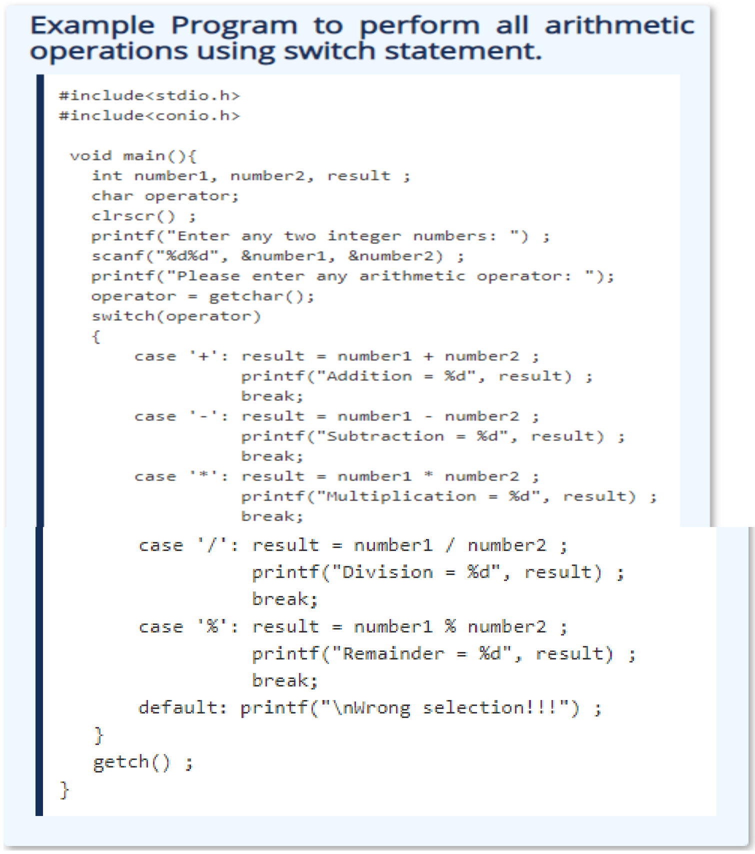
The above three statements do not need any condition to control the program execution flow.

6.6.1 Break statement

In **C**, the **break statement** is used to perform the following two things...

1. **break statement is used to terminate the switch case statement**
2. **break statement is also used to terminate looping statements like while, do-while and for.**

When a **break** statement is encountered inside the switch case statement, the execution control moves out of the switch statement directly. For example, consider the following program.



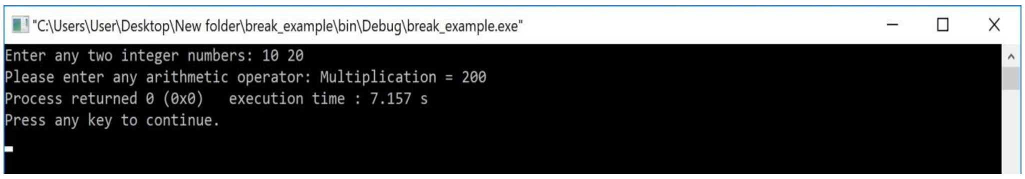
Example Program to perform all arithmetic operations using switch statement.

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

 void main(){
   int number1, number2, result ;
   char operator;
   clrscr() ;
   printf("Enter any two integer numbers: ") ;
   scanf("%d%d", &number1, &number2) ;
   printf("Please enter any arithmetic operator: ");
   operator = getchar();
   switch(operator)
   {
       case '+': result = number1 + number2 ;
                 printf("Addition = %d", result) ;
                 break;
       case '-': result = number1 - number2 ;
                 printf("Subtraction = %d", result) ;
                 break;
       case '*': result = number1 * number2 ;
                 printf("Multiplication = %d", result) ;
                 break;
       case '/': result = number1 / number2 ;
                 printf("Division = %d", result) ;
                 break;
       case '%': result = number1 % number2 ;
                 printf("Remainder = %d", result) ;
                 break;
       default: printf("\nWrong selection!!!") ;
   }
   getch() ;
}
```

```
        case '/': result = number1 / number2 ;
                  printf("Division = %d", result) ;
                  break;
        case '%': result = number1 % number2 ;
                  printf("Remainder = %d", result) ;
                  break;
        default: printf("\nWrong selection!!!") ;
    }
    getch() ;
}
```

```
        case '/': result = number1 / number2 ;
                  printf("Division = %d", result) ;
                  break;
        case '%': result = number1 % number2 ;
                  printf("Remainder = %d", result) ;
                  break;
        default: printf("\nWrong selection!!!") ;
    }
    getch() ;
}
```

Output:

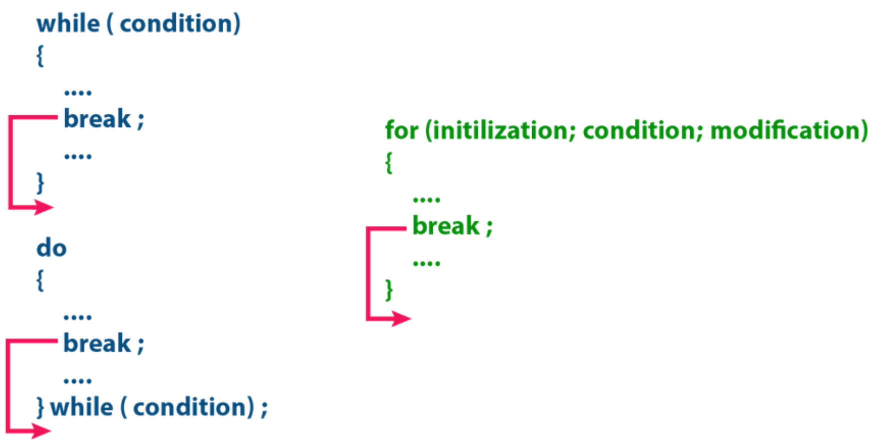


"C:\Users\User\Desktop\New folder\break_example\bin\Debug\break_example.exe"

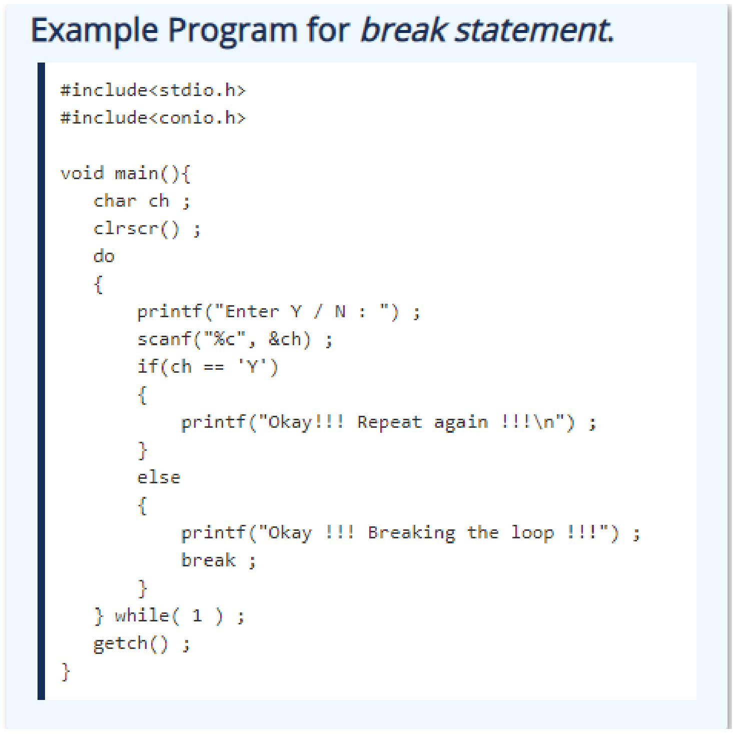
```
Enter any two integer numbers: 10 20
Please enter any arithmetic operator: Multiplication = 200
Process returned 0 (0x0)   execution time : 7.157 s
Press any key to continue.
```

When the **break** statement is encountered inside the looping statement, the execution control moves out of the looping statements.

The **break** statement execution is as shown in the following figure.



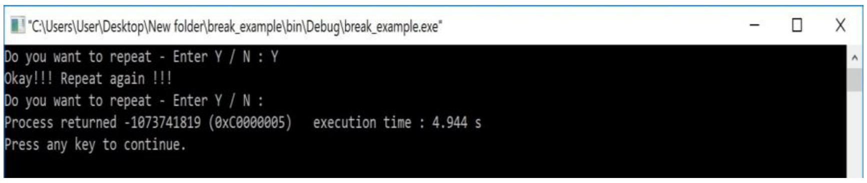
For example, consider the following example program...



Example Program for *break statement.*

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

void main(){
   char ch ;
   clrscr() ;
   do
   {
       printf("Enter Y / N : ") ;
       scanf("%c", &ch) ;
       if(ch == 'Y')
       {
           printf("Okay!!! Repeat again !!!\n") ;
       }
       else
       {
           printf("Okay !!! Breaking the loop !!!") ;
           break ;
       }
   } while( 1 ) ;
   getch() ;
}
```

Output:

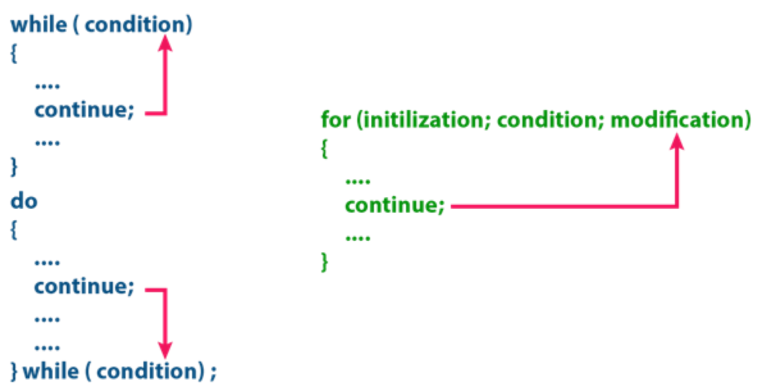


```
"C:\Users\User\Desktop\New folder\break_example\bin\Debug\break_example.exe"
Do you want to repeat - Enter Y / N : Y
Okay!!! Repeat again !!!
Do you want to repeat - Enter Y / N :
Process returned -1073741819 (0xC0000005)   execution time : 4.944 s
Press any key to continue.
```

6.6.2 Continue statement

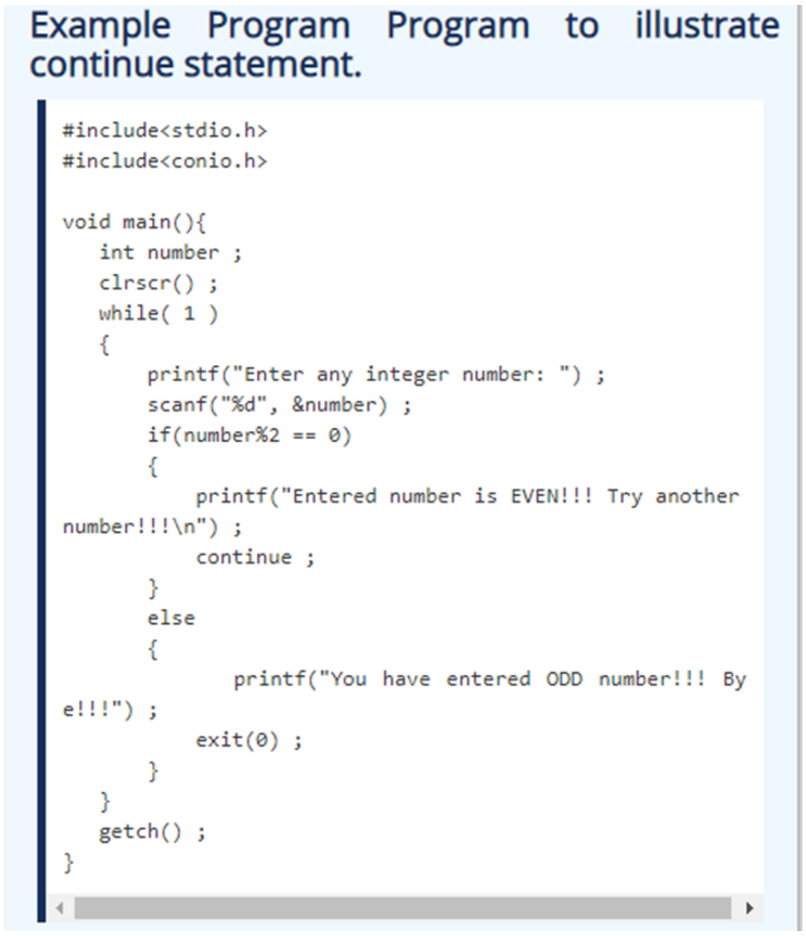
The **continue** statement is used to move the program execution control to the beginning of the looping statement. When the **continue** statement is encountered in a looping statement, the execution control skips the rest of the statements in the looping block and directly jumps to the beginning of the loop. The **continue** statement can be used with looping statements like while, do-while and for.

When we use **continue** statement with **while** and **do-while** statements the execution control directly jumps to the condition. When we use **continue** statement with **for** statement the execution control directly jumps to the modification portion (increment/decrement/any modification) of the for loop. The **continue** statement execution is as shown in the following figure.



```
while ( condition)
{
   ....
   continue;
   ....
}
do
{
   ....
   continue;
   ....
   ....
} while ( condition) ;
```

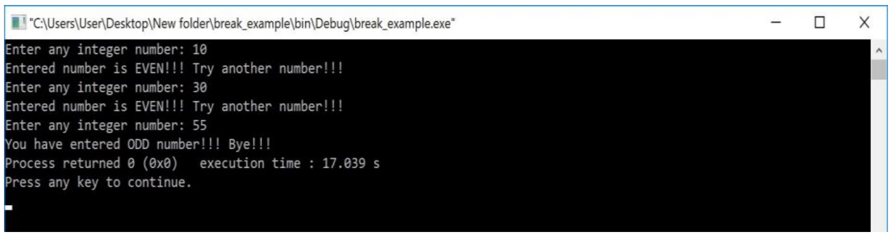
```
for (initilization; condition; modification)
{
   ....
   continue;
   ....
}
```



Example Program Program to illustrate continue statement.

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

void main(){
   int number ;
   clrscr() ;
   while( 1 )
   {
       printf("Enter any integer number: ") ;
       scanf("%d", &number) ;
       if(number%2 == 0)
       {
           printf("Entered number is EVEN!!! Try another
number!!!\n") ;
           continue ;
       }
       else
       {
              printf("You have entered ODD number!!! By
e!!!") ;
           exit(0) ;
       }
   }
   getch() ;
}
```

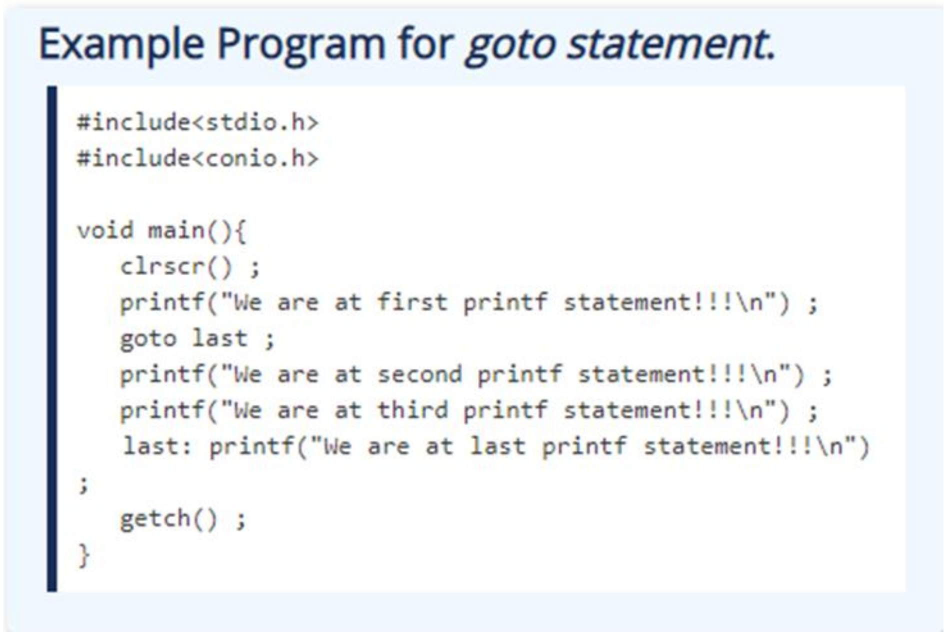
Output:



```
"C:\Users\User\Desktop\New folder\break_example\bin\Debug\break_example.exe"
Enter any integer number: 10
Entered number is EVEN!!! Try another number!!!
Enter any integer number: 30
Entered number is EVEN!!! Try another number!!!
Enter any integer number: 55
You have entered ODD number!!! Bye!!!
Process returned 0 (0x0)   execution time : 17.039 s
Press any key to continue.
```

6.6.3 goto statement

The **goto** statement is used to jump from one line to another line in the program. Using **goto** statement we can jump from top to bottom or bottom to top. To jump from one line to another line, the goto statement requires a **label**. Label is a name given to the instruction or line in the program. When we use a **goto** statement in the program, the execution control directly jumps to the line with the specified label.



Example Program for *goto statement.*

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

void main(){
   clrscr() ;
   printf("We are at first printf statement!!!\n") ;
   goto last ;
   printf("We are at second printf statement!!!\n") ;
   printf("We are at third printf statement!!!\n") ;
   last: printf("We are at last printf statement!!!\n")
;
   getch() ;
}
```

Output:

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use break, continue and goto statements, we must follow the following...

- The **break** is a keyword so it must be used only in lower case letters.
- The **break** statement can not be used with **if** statement.
- The **break** statement can be used only in switch case and looping statements.
- The **break** statement can be used with **if** statement, only if that **if statement** is written inside the switch case or looping statements.
- The **continue** is a keyword so it must be used only in lower case letters.
- The **continue** statement is used only within **looping statements**.
- The **continue** statement can be used with **if** statement, only if that **if statement** is written inside the looping statements.
- The **goto** is a keyword so it must be used only in lower case letters.

- The **goto** statement must require a **label**.
- The **goto** statement can be used with any statement like if, switch, while, do-while, and for, etc.

Check your Progress-2

3. Explain Exit control looping statement

____

____

____

2. Explain goto statement in C

____

____

____

6.7 LET US SUM UP

Branching statement helps to jump from one part of the program to another, these statements help. The control transfer may be unconditional or conditional. A loop statement allows us to execute a statement or group of statements multiple times.

6.8 KEYWORDS

Compares - estimate, measure, or note the similarity or dissimilarity between.

Execution - the carrying out of a plan, order, or course of action.

Modification - a change made.

6.9 QUESTIONS FOR REVIEW

1. Explain switch statement in C

2. Explain looping statement in C

3. Discuss execution using ‘for’ statement

4. Brief about ‘break’ in C

6.10 SUGGESTED READINGS AND REFERENCES

1. Fundamentals of Real Analysis, S K. Berberian, Springer.
2. Measure Theory and Integration, G. De Barra, New Age International Publ.
3. Real Analysis, H. L. Royden.
4. Principles of Mathematical Analysis, W. Rudin.
5. Lectures on Real Analysis, J. Yeh, World Sci.
6. R. G. Bartle, The Elements of Integration, John Wiley & Sons, Inc. New York, 1966

6.11 ANSWERS TO CHECK YOUR PROGRESS

1. Provide important point to remember- 6.2

2. Provide definition– 6.3

3.Provide explanation – 6.4

4. Provide explanation – 6.6.3

UNIT-7 DESIGNING STRUCTURED PROGRAMS IN C

STRUCTURE

7.0 OBJECTIVES

Understand the Functions and types of Functions in C

Enumerate the concept of Parameter Passing in C

7.1 INTRODUCTION

Structured programming is a programming technique in which a larger program is divided into smaller subprograms to make it easy to understand, easy to implement and makes the code reusable, etc. Structured programming enables code reusability. **Code reusability** is a method of writing code once and using it many times. Using a structured programming technique, we write the code once and use it many times. Structured programming also makes the program easy to understand, improves the quality of the program, easy to implement and reduces time.

In C, the structured programming can be designed using **functions** concept. Using functions concept, we can divide the larger program into smaller subprograms and these subprograms are implemented individually. Every subprogram or function in C is executed individually.

7.2 FUNCTIONS IN C

When we write a program to solve a larger problem, we divide that larger problem into smaller subproblems and are solved individually to make the program easier. In C, this concept is implemented using functions. Functions are used to divide a larger program into smaller subprograms such that the program becomes easy to understand and easy to implement. A function is defined as follows...

Function is a subpart of a program used to perform a specific task and is executed individually.

Every function in C has the following...

- **Function Declaration (Function Prototype)**
- **Function Definition**
- **Function Call**

7.2.1 FUNCTION DECLARATION

The function declaration tells the compiler about function name, the data type of the return value and parameters. The function declaration is also called a function prototype. The function declaration is performed before the main function or inside the main function or any other function.

Function declaration syntax -

```
returnType functionName(parametersList);
```

In the above syntax, **returnType** specifies the data type of the value which is sent as a return value from the function definition. The **functionName** is a user-defined name used to identify the function uniquely in the program. The **parametersList** is the data values that are sent to the function definition.

7.2.2FUNCTION DEFINITION

The function definition provides the actual code of that function. The function definition is also known as the **body of the function**. The actual task of the function is implemented in the function definition. That means the actual instructions to be performed by a function are written in function definition. The actual instructions of a function are written inside the braces "{ }". The function definition is performed before the main function or after the main function.

Function definition syntax -

```
returnType functionName(parametersList)
{

Actual code...


}
```

7.2.3 FUNCTION CALL

The function call tells the compiler when to execute the function definition. When a function call is executed, the execution control jumps to the function definition where the actual code gets executed and returns to the same functions call once the execution completes. The function call is performed inside the main function or any other function or inside the function itself.

Function call syntax -

```
functionName(parameters);
```

ADVANTAGES OF FUNCTIONS

- Using funcions we can implement modular programming.
- Functions make the program more readable and understandable.
- Using functions the program implementation becomes easy.
- Once a function is created it can be used many times (**code re-usability**).
- Using functions larger programs can be divided into smaller modules.

7.3 TYPES OF FUNCTIONS IN C

In C Programming Language, based on providing the function definition, functions are divided into two types. Those are as follows...

- **System Defined Functions**
- **User Defined Functions**

7.3.1 System Defined Functions

The C Programming Language provides pre-defined functions to make programming easy. These pre-defined functions are known as syatem defined functions. The system defined function is defined as follows...

The function whose definition is defined by the system is called as system defined function.

The system defined functions are also called as **Library Functions** or **Standard Functions** or **Pre-Defined Functions**. The implementation of system defined functions is already defined by the system.

In C, all the system defined functions are defined inside the **header files** like **stdio.h**, **conio.h**, **math.h**, **string.h** etc.,

For example, the funtions **printf()** and **scanf()** are defined in the header file called **stdio.h**.

Whenever we use system defined functions in the program, we must include the respective header file using **#include** statement. For example, if we use a system defined function **sqrt()** in the program, we must include the header file called **math.h** because the function **sqrt()** is defined in **math.h**.

Points to be Remembered

- System defined functions are declared in header files
- System defined functions are implemented in .dll files. (DLL stands for **Dynamic Link Library**).
- To use system defined functions the respective header file must be included.

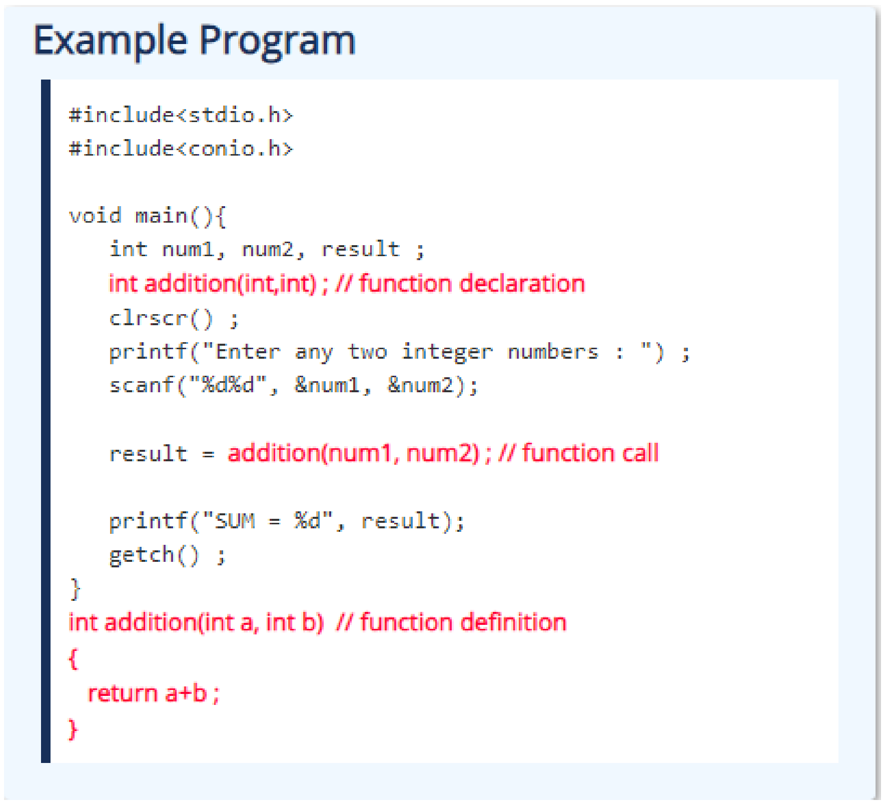
7.3.2 USER DEFINED FUNCTIONS

In C programming language, users can also create their own functions. The functions that are created by users are called as user defined functions. The user defined function is defined as follows...

The function whose definition is defined by the user is called as user defined function.

That means the function that is implemented by user is called as user defined function. For example, the function **main** is implemented by user so it is called as user defined function.

In **C** every user defined function must be declared and implemented. Whenever we make function call the function definition gets executed. For example, consider the following program in which we create a function called **addition** with two paramenters and a return value.



Example Program

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

void main(){
   int num1, num2, result ;
   int addition(int,int) ; // function declaration
   clrscr() ;
   printf("Enter any two integer numbers : ") ;
   scanf("%d%d", &num1, &num2);

   result = addition(num1, num2) ; // function call

   printf("SUM = %d", result);
   getch() ;
}
int addition(int a, int b)  // function definition
{
  return a+b ;
}
```

In the above example program, the function declaration statement **"int addition(int,int)"** tells the compiler that there is a function with name **addition** which takes two integer values as parameters and returns an integer value. The function call statement takes the execution control to the **additon()** definition along with values of **num1** and **num2**. Then **function definition** executes the code written inside it and comes back to the **function call** along with **return value**.

In the concept of functions, the function call is known as **"Calling Function"** and the function definition is known as **"Called Function"**

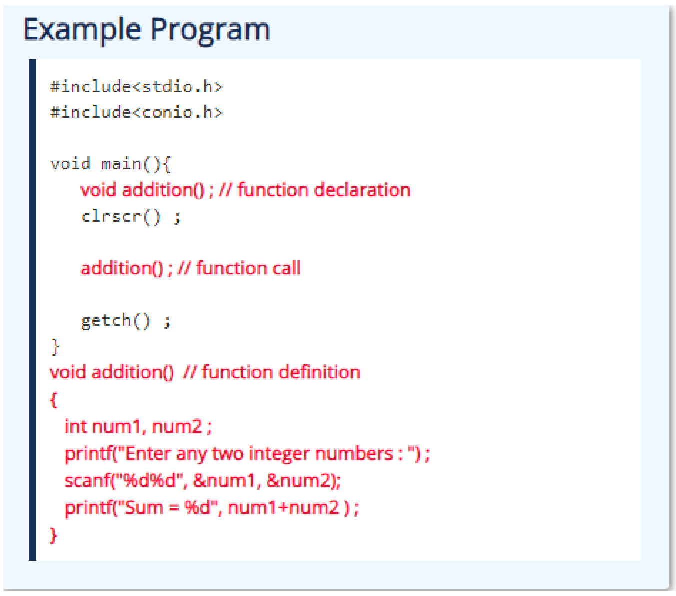
When we make a function call, the execution control jumps from calling function to called function. After executing the called function, the execution control comes back to calling function from called function. When the control jumps from calling function to called function it may carry one or more data values called "**Paramenters**" and while coming back it may carry a single value called "**return value**". That means the

data values transferred from calling function to called function are called as **Parameters** and the data value transferred from called funcion to calling function is called **Return value**.
Based on the data flow between the calling function and called function, the functions are classified as follows...

- **Function without Parameters and without Return value**
- **Function with Parameters and without Return value**
- **Function without Parameters and with Return value**
- **Function with Parameters and with Return value**

Function without Parameters and without Return value

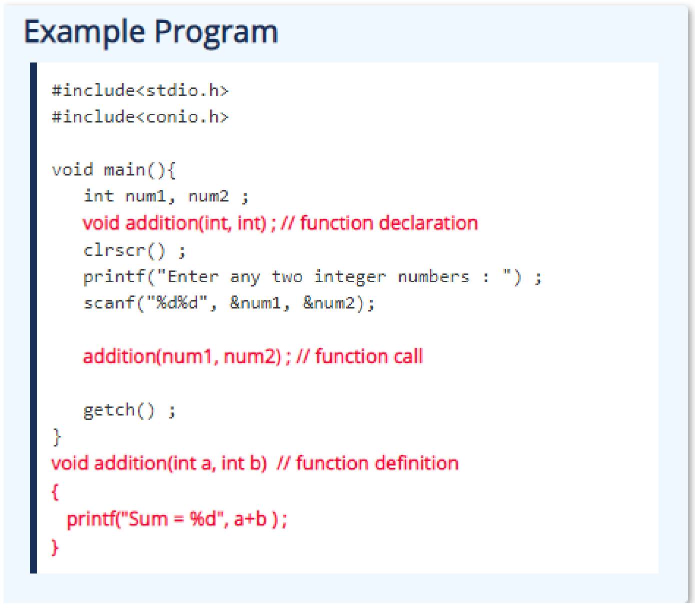
In this type of functions there is no data transfer between calling function and called function. Simply the execution control jumps from calling-function to called function and executes called function, and finally comes back to the calling function. For example, consider the following program...


Example Program

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

void main(){
    void addition() ; // function declaration
    clrscr() ;

    addition() ; // function call

    getch() ;
}
void addition()  // function definition
{
  int num1, num2 ;
  printf("Enter any two integer numbers : ") ;
  scanf("%d%d", &num1, &num2);
  printf("Sum = %d", num1+num2 ) ;
}
```

Function with Parameters and without Return value

In this type of functions there is data transfer from calling-function to called function (parameters) but there is no data transfer from called function to calling-function (return value). The execution control jumps from calling-function to called function along with the parameters and executes called function, and finally comes back to the calling function. For example, consider the following program...

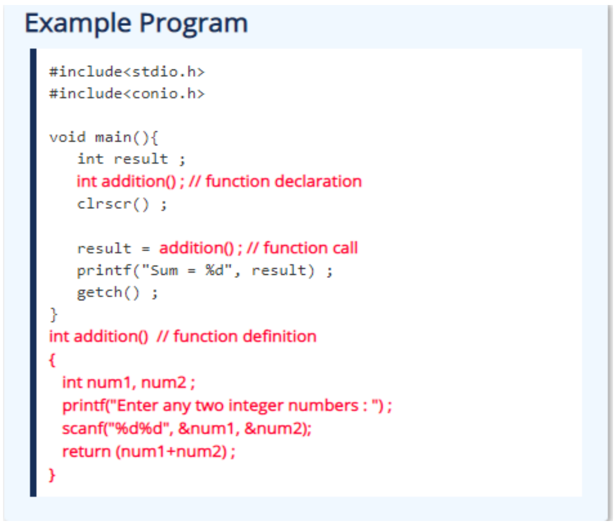


Example Program

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

void main(){
   int num1, num2 ;
   void addition(int, int) ; // function declaration
   clrscr() ;
   printf("Enter any two integer numbers : ") ;
   scanf("%d%d", &num1, &num2);

   addition(num1, num2) ; // function call

   getch() ;
}
void addition(int a, int b)  // function definition
{
  printf("Sum = %d", a+b ) ;
}
```

Function without Parameters and with Return value

In this type of functions there is no data transfer from calling-function to called-function (parameters) but there is data transfer from called function to calling-function (return value). The execution control jumps from calling-function to called function and executes called function, and finally comes back to the calling function along with a return value. For example, consider the following program...

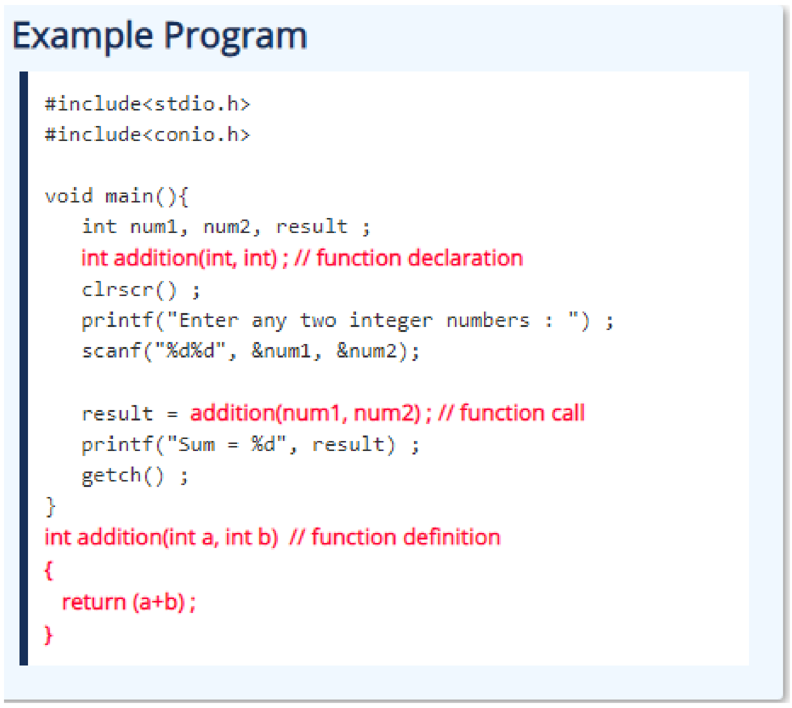


Example Program

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

void main(){
   int result ;
   int addition() ; // function declaration
   clrscr() ;

   result = addition() ; // function call
   printf("Sum = %d", result) ;
   getch() ;
}
int addition()  // function definition
{
  int num1, num2 ;
  printf("Enter any two integer numbers : ") ;
  scanf("%d%d", &num1, &num2);
  return (num1+num2) ;
}
```

Function with Parameters and with Return value

In this type of functions there is data transfer from calling-function to called-function (parameters) and also from called function to calling-function (return value). The execution control jumps from calling-function to called function along with parameters and executes called function, and finally comes back to the calling function along with a return value. For example, consider the following program...



Example Program

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

void main(){
   int num1, num2, result ;
   int addition(int, int) ; // function declaration
   clrscr() ;
   printf("Enter any two integer numbers : ") ;
   scanf("%d%d", &num1, &num2);

   result = addition(num1, num2) ; // function call
   printf("Sum = %d", result) ;
   getch() ;
}
int addition(int a, int b)  // function definition
{
  return (a+b) ;
}
```

Points to be Remembered

- The parameters specified in calling function are said to be **Actual Parameters**.
- The parameters declared in called function are said to be **Formal Parameters**.
- The value of actual parameters is always copied into formal parameters.

Check your Progress-1

1. Define Function and state its advantage

____

____

____

2. What is system defined function

____

____

____

3. Define parameters and return value

____

____

____

7.4 PARAMETER PASSING IN C

When a function gets executed in the program, the execution control is transferred from calling-function to called function and executes function definition, and finally comes back to the calling function. When the execution control is transferred from calling-function to called-function it may carry one or number of data values. These data values are called as **parameters**.

Parameters are the data values that are passed from calling function to called function.

In C, there are two types of parameters and they are as follows...

- **Actual Parameters**
- **Formal Parameters**

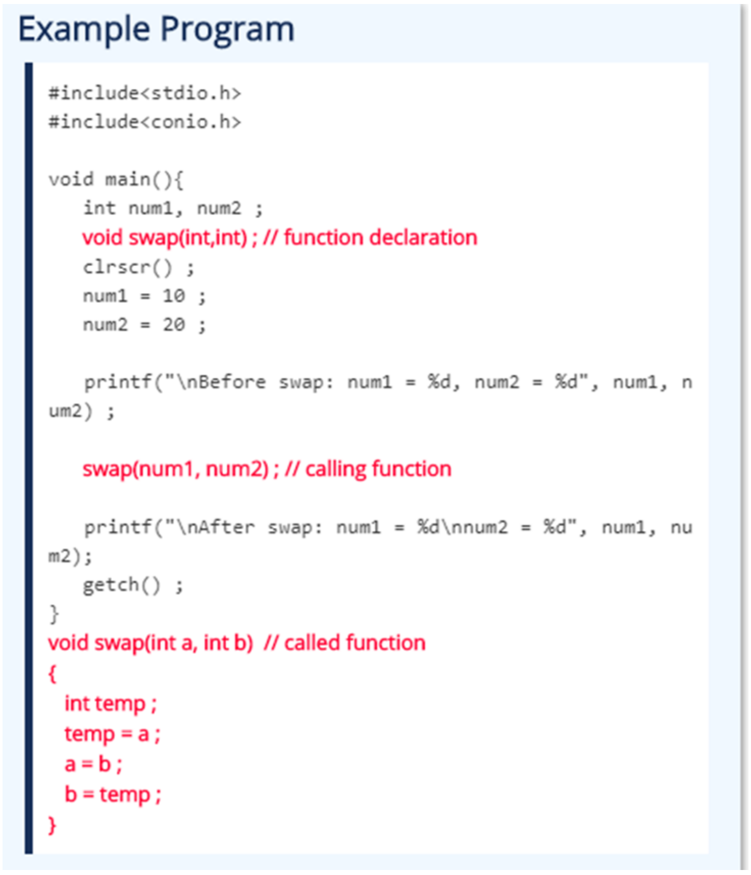
The **actual parameters** are the parameters that are specified in calling function. The **formal parameters** are the parameters that are declared at called function. When a function gets executed, the copy of actual parameter values are copied into formal parameters.

In C Programming Language, there are two methods to pass parameters from calling function to called function and they are as follows...

- **Call by Value**
- **Call by Reference**

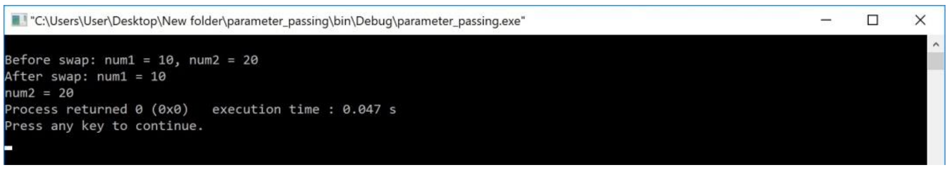
7.4.1 CALL BY VALUE

In **call by value** parameter passing method, the copy of actual parameter values are copied to formal parameters and these formal parameters are used in called function. The changes made on the formal parameters does not effect the values of actual parameters**.** That means, after the execution control comes back to the calling function, the actual parameter values remains same. For example consider the following program...



```
Example Program

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

void main(){
   int num1, num2 ;
   void swap(int,int) ; // function declaration
   clrscr() ;
   num1 = 10 ;
   num2 = 20 ;

   printf("\nBefore swap: num1 = %d, num2 = %d", num1, n
um2) ;

   swap(num1, num2) ; // calling function

   printf("\nAfter swap: num1 = %d\nnum2 = %d", num1, nu
m2);
   getch() ;
}
void swap(int a, int b)  // called function
{
  int temp ;
  temp = a ;
  a = b ;
  b = temp ;
}
```

Output:


"C:\Users\User\Desktop\New folder\parameter_passing\bin\Debug\parameter_passing.exe"

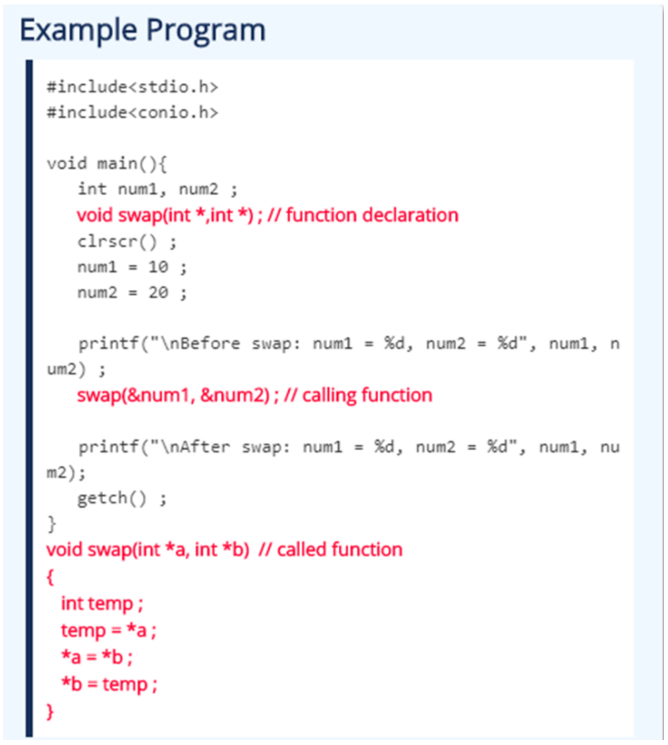
```
Before swap: num1 = 10, num2 = 20
After swap: num1 = 10
num2 = 20
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

In the above example program, the variables **num1** and **num2** are called actual parameters and the variables **a** and **b** are called formal parameters. The value of **num1** is copied into **a** and the value of num2 is copied into **b**. The changes made on variables **a** and **b** does not affect the values of **num1** and **num2**.

7.4.2 CALL BY REFERENCE

In **Call by Reference** parameter passing method, the memory location address of the actual parameters is copied to formal parameters. This address is used to access the memory locations of the actual parameters in called function. In this method of parameter passing, the formal parameters must be **pointer** variables.

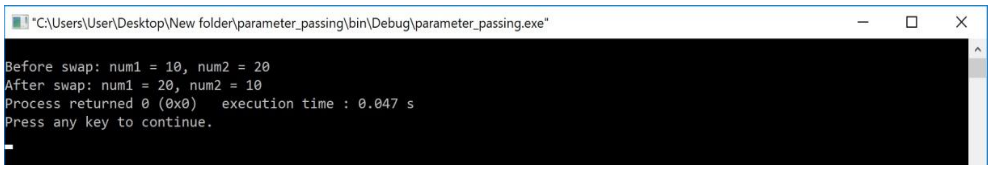
That means in call by reference parameter passing method, the address of the actual parameters is passed to the called function and is recieved by the formal parameters (pointers). Whenever we use these formal parameters in called function, they directly access the memory locations of actual parameters. So **the changes made on the formal parameters effects the values of actual parameters**. For example consider the following program...



Example Program

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

void main(){
   int num1, num2 ;
   void swap(int *,int *) ; // function declaration
   clrscr() ;
   num1 = 10 ;
   num2 = 20 ;

   printf("\nBefore swap: num1 = %d, num2 = %d", num1, n
um2) ;
   swap(&num1, &num2) ; // calling function

   printf("\nAfter swap: num1 = %d, num2 = %d", num1, nu
m2);
   getch() ;
}
void swap(int *a, int *b)  // called function
{
  int temp ;
  temp = *a ;
  *a = *b ;
  *b = temp ;
}
```

Output:



```
"C:\Users\User\Desktop\New folder\parameter_passing\bin\Debug\parameter_passing.exe"

Before swap: num1 = 10, num2 = 20
After swap: num1 = 20, num2 = 10
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

In the above example program, the addresses of variables **num1** and **num2** are copied to pointer variables **a** and **b**. The changes made on the pointer variables **a** and **b** in called function effects the values of actual parameters **num1** and **num2** in calling function.

Check your Progress-2

4. Define Actual parameters.

____
____
____

5. What is ‘Call by Value’?

____
____
____

7.5 LET US SUM UP

A structured programming technique, we write the code once and use it many times. Structured programming also makes the program easy to understand, improves the quality of the program, easy to implement and reduces time.

7.6 KEYWORDS

Parameters - a numerical or other measurable factor forming one of a set that defines a system or sets the conditions of its operation.

Defined - having a definite outline or specification; precisely marked or stated.

Implemented - put (a decision, plan, agreement, etc.) into effect

7.7 QUESTIONS FOR REVIEW

1. Explain types of function in C

2. Explain user defined function in C

3. Describe the types of parameters

4. Explain ‘Call by reference’.

7.8 SUGGESTED READINGS AND REFERENCES

- B. Gottfried: Programming with C , Tata McGraw-Hill Edition 2002.
- E. Balagurusamy : Programming in ANSI C, Tata Mcgraw Hill - Edition 2002.
- Brain W. Kernighan & Dennis M. Ritchie, The C Programme Language, 2nd Edition (ANSI features) , Prentice Hall 1989.
- Let Us C- Y.P. Kanetkar, BPB Publication - 2002.
- Analysis of Numerical Methods—Isacsons& Keller.
- Numerical solutions of Ord. Diff. Equations—M K Jain
- Numerical solutions of Partial Diff. Equations—G D Smith.
- Programming with C, B. Gottfried, Tata-McGraw Hill
- Programming with C, K. R. Venugopal and Sudeep R. Prasad, Tata-McGraw Hill

7.9 ANSWERS TO CHECK YOUR PROGRESS

1. Provide explanation, syntax and advantage- 7.2

2. Provide definition– 7.3.1

3.Provide definition – 7.3.2

4. Provide definition – 7.4

5. Provide explanation – 7.4.1