

C in Hindi



BccFalna.com
097994-55505

Kuldeep Chand

Learn Programming Fundamentals with Deep Details in easy to understand Hindi Language.

So many Example Programs and Code Fragements to easily understand the Programming Concept.

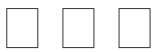
Detaild Program Flow Discussion to understand the working of the Program Step by Step.

Without learning “C” Language, you can’t learn any Modern Programming Language which is used mostly for Professional Application Software development like Java or C#.

So, learn “C” and start moving in the way of Professional Development for full of Joy and Healthy Programming Career.

C

In Hindi



Kuldeep Chand

BetaLab Computer Center
Falna

Programming Language “C” in HINDI

Copyright © 2011 by Kuldeep Chand

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editors: **Kuldeep Chand**

Distributed to the book trade worldwide by Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

e-mail bccfalna@gmail.com,

or

visit <http://www.bccfalna.com>

For information on translations, please contact BetaLab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

Phone **097994-55505**

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, the author shall not have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this book.

**This book is dedicated to those
who really wants to be
a
PROFESSIONAL DEVELOPER**

**INDEX
OF
CONTENTS**

Table of Contents

Introduction	12
Data – Value or a Set of Values	12
Processing – Generating Results	13
Information – Processed Data	13
What is a Computer	14
System – Group of Units to Solve a Problem	14
Program and Software.....	14
System Software:.....	15
Application Software:	15
Computer Architecture.....	15
I/O Devices	16
Center Processing Unit (CPU)	16
Memory	17
Types of Programming	17
Hardware Programming.....	18
Software Programming	18
Language Introduction	20
Level of Computer Languages	20
Low Level Language or Machine Language	20
Middle Level or Assembly Language	20
High Level Language	20
Assembler.....	21
Compiler and Interpreter.....	21
Similarities between Real Word and Computer Program	21
Steps of Program	22
Characteristics of a Good Program	23
Problem – Doing Something.....	23
Algorithm – List of Sequential Steps to Solve a Problem	24
History of Programming Language “C”	25
Characteristics of “C”	25
Layout Structure of “C” Programs	26
Coding Structure of “C” Programs	27
Functions – Pre-Defined and User-Defined.....	30
Input Section	31
Process Section	32
Output Section	32
Output Function	32
Basic Elements of “C”	38
“C” Characterset	38
“C” Tokens.....	38
Keywords या Reserve Words.....	39
Identifiers – Constant and Variable Name	39
Constants and Variables.....	41
Identifier Declaration	44
Data and Data Types.....	47
Integer.....	48
Float.....	51
Double.....	51
Character	52
Data Types Modifiers	53

Control String	54
Preprocessor Directive.....	58
Literal	63
Types of Instructions	111
Type Declaration Instruction	111
Arithmetical Instruction.....	112
Control Instruction.....	115
Precedence of Operators	116
Type Conversion in Expressions	118
Automatic Type Conversion.....	119
Manual Type Conversion OR Casting	119
Function Calling and Function Arguments.....	121
String and Character Functions	122
Working with String.....	122
Working with Characters.....	128
Formatted Input.....	132
Formatted Output.....	136
Control Statement and Looping	146
Program Control	146
Types Of Control Statement.....	146
Sequential Statements.....	146
Conditional Statements.....	147
Iterative Statements	147
Compound Statement or Statement Block	148
if statement.....	148
if – else statement	151
Nested if else statement.....	153
if – else if – else Ladder statement.....	157
switch statement.....	160
goto Statement	163
Looping Statements.....	166
for Loop.....	166
Nesting of Loop.....	174
while Loop.....	178
Do...while Loop.....	181
break Statement	182
continue Statement.....	182
Arrays.....	190
Linear Arrays	193
2-D Array	200
Initializing Value of a Character Array (String)	202
Functions.....	208
Library Functions	208
User Defined Functions	208
Calling Function and Called Function.....	209
Function Definition.....	209
Argument Variables Declaration.....	210
Local Variables	210
Return (Expression).....	210
Statement Block.....	210
Function Prototype.....	210
Types of Functions	211

Function Without Argument And Return Value	211
Function With Argument But No Return Value	215
Function With Argument And Return Value	221
Function Without Argument But Return Value	224
Recursion and Recursive Function	225
Storage Classes	229
Type of Variables In Program	229
Automatic Storage Class	231
Extern Storage Class	234
Static Storage Class	236
Register Storage Class	237
Pointers	240
Understanding Pointers	241
Defining Pointers	242
Accessing the Address of the Variable	243
Accessing a Address Through It's Pointer	244
Pointer Expressions.....	246
Addition and Subtraction A Number to a Pointer	247
Pointer Increment and Scale Factor	247
Function with Arrays	249
strcat() Function	251
strcpy() Function	253
strlen() Function	253
strcom() Function	254
Working with Binary Digits	255
Subtraction One Pointer to another Pointer	258
Comparison of two Pointers	258
Array in Function through Pointer	259
Function Returning Pointers	261
One – Dimensional Array with Pointer	262
Pointer with 2-Dimensional Array	266
Array of Pointers	269
Array of Pointers To String	271
C Preprocessor	284
Directives	284
Macro Substitution Directive	285
File Inclusion Directive.....	291
Conditional Compilations.....	292
Function And Macros.....	296
Build Process.....	297
Dynamic Memory Allocation.....	299
malloc() Function	300
calloc () Function	303
free() Function	304
realloc() Function	305
Structure.....	308
Structure Definition	308
Structure Declaration	309
Accessing the Structure Members	310
Initializing the Structure Members	311
Structure with Array	311
Array within Structure	314

Structure Within Structure (Nested Structure)	315
Structure with Function	320
Union	325
Pointers and Structure	327
Typedef.....	331
Enumerated Data Type	332
Bit Fields.....	334
File Management in C	339
Opening a File	339
File Opening Modes	341
getc()	342
putc()	342
getw ().....	347
putw ().....	348
feof()	348
fgets ()	349
fputs ()	349
fprintf()	350
fscanf().....	350
Standard DOS Services	352
rewind();.....	356
ferror();.....	356
fseek();.....	357
ftell();.....	358
Command Line Argument.....	365
Low Level Disk I/O.....	369
Operating System AND Windows Programming.....	377
DOS Programming Model	377
Event	378
Hardware Event	379
Software Event.....	379
Interface.....	379
Programming in DOS vs Windows	380
Windows Programming Model - Message Passing System	380
Device Driver	383
Dynamic Linked Library (DLL) Files	383
USER32.DLL	384
GDI32.DLL	384
KERNEL32.DLL	384
WINMM32.DLL	384
What Are Messages	385
Event – Driven Architecture.....	385
Traditional MS DOS Program Model	386
Windows Program Model.....	386
Windows Application Development Tools	388
SDK (Software Development Kit)	388
Difference Between Procedure (Routine) And Functions	389
Message Passing	389
Handles	390
Windows Programming In C	393
Main Program	393
Hungarian Notation.....	394

Window Class Structure	395
What is a 'Window Class'?	403
Creating and Displaying a Windows	405
Message Queue	415
Message Processing Loop	416
A Complete main Program	420
Window Procedure	424
Complete Window Program.....	431
Text and Graphics in Windows	434
Texts In GUI	434
Painting.....	434
Client Region	435
Non – Client Region.....	435
Invalidate	436
Device Contexts	437
Rectangles.....	443
RECT Structure.....	444
PAINTSTRUCT Structure	445
Device Context (DC) Attributes	446
Painting Text in the Client Area	447
Graphics In GUI.....	448
Types of Graphic Objects You Can Draw in Windows.....	449
Facilities That the Windows GDI Provides	450
Windows RGB Color Format.....	456
Window Origin and Viewport Origin	458
Line Drawing Under Windows.....	461
Background Mode and Color for Lines	468
Drawing Rectangles and Filing with Color	470
Drawing Ellipse	471
Window Graphics – Icon and Menu	475
Message Loop Again.....	480
Resources	482
Menus and Icons	485
WM_CREATE Message	493
Menu Bars and Menus.....	493
Menu Handles.....	494
CreateMenu().....	495
CreatePopupMenu()	495
AppendMenu()	495
LoadImage().....	500
WM_COMMAND	503
Window Graphics and Dialog Box.....	510
Modal Dialog Box	510
DialogBox Macro.....	513
Dialog Properties Dialog Box	522
Modeless Dialog Box.....	528
Controls	534
BUTTON	535
COMBOBOX.....	535
EDIT.....	535
LISTBOX.....	535
SCROLLBAR	536

STATIC	536
Messages	536
Window Graphics and Dialog Box Resource	545
GDI	545
Device Contexts	545
Bitmaps.....	546
GDI Leaks.....	546
Displaying Bitmaps	547
Getting The Window DC	549
Setting up a Memory DC for the Bitmap	550
Drawing	550
Cleanup.....	553
Messages	558
Event-Driven Input Versus "Hurry Up and Wait".....	558
Focus	559
Caret.....	560
Keyboard Messages.....	564
Mouse Messages	570
Timer Messages	579

PROGRAMMING INTRODUCTION

Introduction

सभ्यता की शुरुआत से ही मानव को Information की जरूरत रही है। इसीलिए वह समय-समय पर सूचनाओं को एकत्रित करने व उन सूचनाओं के आधार पर सही व उचित निर्णय लेने के नए व विकसित तरीके खोजता रहा है। सूचना की आवश्यकता व महत्व के कारण सबसे पहला आविष्कार कागज व कलम हुआ।

जैसे-जैसे मानव का विकास होता गया वैसे-वैसे उसने नए शहर, राज्य व देश बनाए और उन देशों के बीच व्यापार व वाणिज्य के कारण विभिन्न सम्बंध बने और आज केवल व्यापार व वाणिज्य ही नहीं बल्कि जीवन की लगभग हर सूचना का Internet के माध्यम से इन देशों के बीच आदान प्रदान हो रहा है। कृषि क्रांति व औद्योगिक क्रांति के बाद आज हम सूचना क्रांति के युग में जी रहे हैं।

पहले सूचनाओं को मिट्टी के बर्तनों पर चित्रात्मक रूप में व शब्दों के रूप में लिखा जाता था। फिर कागज व कलम के विकास से इन पर विभिन्न सूचनाओं को Store करके रखा जाने लगा और आज हम इन्हीं सूचनाओं को Computer पर Manage करते हैं।

विभिन्न प्रकार के आंकड़ों (Data) का संकलन (Collection) करना और फिर उन आंकड़ों को विभिन्न प्रकार से वर्गीकृत (Classify) करके उनका विश्लेषण (Analyze) करना तथा उचित समय पर उचित निर्णय लेने की क्षमता प्राप्त करना, इस पूरी प्रक्रिया को Computer की भाषा में Data Processing करना कहा जाता है।

Data – Value or a Set of Values

असिद्ध तथ्य (Facts) अंक (Figures) व सांख्यिकी (Statics) का वह समूह, जिस पर प्रक्रिया (Processing) करने पर, एक अर्थपूर्ण (Meaningful) सूचना (Information) प्राप्त (Generate) हो, Data कहलाता है। Data, मान या मानों का एक समूह (Value or a Set of Values) होता है, जिसके आधार पर (After Processing) हम निर्णय (Decision) लेते हैं।

इसे एक उदाहरण द्वारा समझने की कोशिश करते हैं। संख्याएं (0 से 9 तक) कुल दस ही होती हैं। लेकिन यदि इन्हें एक व्यवस्थित क्रम में रख दिया जाए, तो एक सूचना Generate होती है। इसलिए ये संख्याएं Data हैं।

अंग्रेजी भाषा में Small व Capital Letters के कुल 52 Characters ही होते हैं, लेकिन यदि इन्हें एक सुव्यवस्थित क्रम में रखा जाए, तो हजारों पुस्तकें बन सकती हैं। इसलिए ये Characters Data हैं।

Computer में हम इन्हीं दो रूपों में वास्तविक जीवन की विभिन्न बातों को Store करते हैं और उन पर Processing करके आवश्यकतानुसार Information Generate करते हैं। जैसे किसी School के विभिन्न Students की ये जानकारी Manage करनी हो कि किसी Class में कौन-कौन से Students हैं, उनका Serial Number क्या है और वे किस Address पर रहते हैं, तो ये सभी तथ्य असिद्ध रूप में Computer के लिए Data हैं क्योंकि किसी Student के Serial number को 0 से 9 के कुछ अंकों के समूह रूप में Express किया जाता है और Student का नाम व पता Characters के एक सुव्यवस्थित समूह के रूप में Express किया जाता है।

जब 0 से 9 तक के कुछ अंकों को एक समूह में व्यवस्थित किया जाता है तब किसी एक Student का एक Serial Number बन जाता है और जब विभिन्न Characters को एक समूह में व्यवस्थित

किया जाता है, तब किसी Student का नाम व Address बन जाता है। ये नाम व Address ही किसी Student की कुछ **Information** प्रदान करते हैं।

Processing – Generating Results

Data जैसे कि अक्षर, अंक, सांख्यिकी Statics या किसी चित्र को सुव्यवस्थित करना या उनकी Calculation करना, **Processing** कहलाता है। किसी भी Processing में निम्न काम होते हैं:

Calculation	किसी मान को जोड़ना, घटाना, गुणा करना, भाग देना आदि।
Comparison	कोई मान बड़ा, छोटा, शून्य, Positive, Negative, बराबर है, आदि।
Decision Making	किसी Condition के आधार पर निर्णय लेना।
Logic	आवश्यक परिणाम को प्राप्त करने के लिए अपनाया जाने वाला Steps का क्रम।

केवल अंकों की गणना करना ही Processing नहीं कहलाता है। बल्कि किसी भी प्रकार के मान को जैसे कि किसी Document में से गलतियों को खोजने की प्रक्रिया या कुछ नामों के समूह को आरोही (**Ascending**) या अवरोही (**Descending**) क्रम में व्यवस्थित करने की प्रक्रिया को भी Processing की कहते हैं।

Computer में Keyboard से जो भी Data Input किया जाता है, उस Data का तब तक कोई अर्थ नहीं होता है, जब तक कि Computer द्वारा उस Data पर किसी प्रकार की कोई Processing ना की जाए। जैसे उदाहरण के लिए Computer में R, a, d, h, a ये पांच अक्षर अलग-अलग Input किए जाते हैं इसलिए ये सभी अक्षर **Row Data** के समान हैं। Computer इन पांचों अक्षरों पर **Processing** करके इन्हें एक क्रम में व्यवस्थित कर देता है और हमें “**Radha**” नाम प्रदान करता है जो कि एक अर्थपूर्ण सूचना (**Information**) है।

Information – Processed Data

जिस Data पर Processing हो चुकी होती है, उसे **Processed Data** या **Information** कहते हैं। दूसरे शब्दों में कहें तो किसी Data पर Processing होने के बाद जो अर्थपूर्ण परिणाम (**Result**) प्राप्त होता है, उसे ही सूचना (**Information**) कहते हैं। एक Processing से Generate होने वाली किसी Information को हम किसी दूसरी Processing में फिर से Data के रूप में उपयोग में लेकर नई Information Generate कर सकते हैं और ये क्रम आगे भी जारी रखा जा सकता है।

उदाहरण के लिए R, a, m, K, i, l, e, d, R, a, v, a, n ये Characters हम अलग-अलग Input करते हैं। Computer पहले इन पर Processing करके Ram, Killed, व Ravan तीन शब्द बनाता है, जो कि हमारे लिए तीन अलग सूचनाओं को Represent करता है। क्योंकि **Ram, Ravan** व **Killed** तीनों ही शब्द अपने आप में परिपूर्ण हैं, इसलिए ये तीनों ही शब्द एक प्रकार की सूचना हैं जबकि यदि “**Ram Killed Ravan**” लिखा जाए तो इस वाक्य के लिए ये तीनों ही शब्द एक Data के समान हैं, जो Processing के कारण आपस में एक व्यवस्थित क्रम में Arrange होकर एक सूचना प्रदान करते हैं।

सारांश में कहें तो Computer में हम सभी प्रकार की सूचनाओं को Data के आधार पर Store करते हैं। इन Data पर Processing करते हैं जिससे सूचनाएं Generate होती हैं और इन सूचनाओं के आधार पर हम निर्णय लेते हैं। Data वास्तव में कोई अंक अक्षर या चित्र हो सकता है। Computer में इन्हीं मानों को Manage किया जाता है। यानी Data वास्तव में कोई मान या मानों का एक समूह होता है।

What is a Computer

Computer एक ऐसी Electronic Machine है, जो निर्देशों के समूह (जिसे **Program** कहते हैं) के नियंत्रण में Data या तथ्यों पर Processing करके **Information Generate** करता है।

Computer में Data को Accept करने और उस Data पर Required Processing करने के लिए किसी Program को Execute करने की क्षमता होती है। ये किसी Data पर Mathematical व Logical क्रियाएं करने में सक्षम होता है। Computer में Data को Accept करने के लिए Input Devices होती है, जबकि Processed Data यानी Information को प्रस्तुत करने के लिए Output Devices होती हैं। Data पर Processing का काम जिस Device द्वारा सम्पन्न होता है, उसे Central Processing Unit या CPU कहते हैं। ये एक Microprocessor होता है, जिसे Computer का दिमाग भी कहते हैं। किसी भी Computer की निम्नलिखित क्षमताएं होती हैं:

- 1 User द्वारा Supplied Data को Accept कर सकता है।
- 2 Input किए गए Data को Computer की Memory में Store करके Required परिणाम प्राप्त करने के लिए किसी Instructions के समूह यानी किसी Program को Execute कर सकता है, जो कि उस Input किए गए Data पर Processing कर सकता है।
- 3 Data पर Mathematical व Logical क्रियाओं (Operations) को क्रियान्वित (Perform) कर सकता है।
- 4 User की आवश्यकतानुसार Output प्रदान कर सकता है।

System – Group of Units to Solve a Problem

Computer एक System होता है। जब किसी एक या एक से अधिक समस्याओं को सुलझाने या किसी लक्ष्य को प्राप्त करने के लिए कई स्वतंत्र इकाईयां (Individual Units) मिलकर काम कर रहे होते हैं, तो उन इकाईयों के समूह को **System** कहा जाता है।

जैसे कोई Hospital एक System होता है जिसे **Hospital System** कहा जाता है। Doctors, Nurses, चिकित्सा से सम्बंधित विभिन्न उपकरण, Operation Theater, Patient आदि किसी Hospital System की विभिन्न इकाईयां हैं। यदि इन में से किसी की भी कमी हो तो Hospital अधूरा होता है। इसी तरह से Computer भी एक System है, जिसके विभिन्न अवयव जैसे कि Monitor, Mouse, Keyboard, CPU आदि होते हैं और ये सभी आपस में मिलकर किसी समस्या का एक उचित समाधान प्रदान करते हैं।

Program and Software

Computer Programming समझने से पहले हमें ये समझना होता है कि Computer क्या काम करता है और कैसे काम करता है। कम्प्यूटर का मुख्य काम Data का Management करना होता है। हमारे आस-पास जो भी चीजें हमें दिखाई देती हैं, Computer के लिए वे सभी Data हैं और एक Programmer को इन सभी चीजों को Computer में Data के रूप में ही Represent करना होता है। Computer केवल Electrical Signals या मशीनी भाषा को समझता है। ये मशीनी भाषा बाइनरी रूप में होती है, जहां किसी Signal के होने को 1 व ना होने को 0 से प्रदर्शित किया जाता है। यदि हम हमारी किसी बात को Binary Format में Computer में Feed कर सकें, तो Computer हमारी बात को समझ सकता है।

Computer भाषा वह भाषा होती है जिसे Computer समझ सकता है, क्योंकि हर Computer भाषा का एक Software होता है। ये Software हमारी बात को Computer के समझने योग्य मशीनी भाषा या Binary Format में Convert करता है। Computer को कोई बात समझाने के लिए उसे एक निश्चित क्रम में सूचनाएं देनी होती हैं, जिन्हें **Instructions** कहा जाता है।

जब किसी काम का एक सुव्यवस्थित परिणाम प्राप्त करने के लिए Computer को दिए जाने वाले विभिन्न प्रकार के Instructions को एक समूह के रूप में व्यवस्थित कर दिया जाता है, तो Instructions के इस समूह को **Program** कहा जाता है। Computer इन दी गई Instructions के अनुसार काम करता है और जिस तरह का परिणाम प्राप्त करने के लिए Program लिखा गया होता है, Computer हमें Program के आधार पर उसी प्रकार का परिणाम प्रदान कर देता है।

Computer में हर Electrical Signal या उसके समूह को Store करके रखने की सुविधा होती है। इन Electrical Signals के समूह को **File** कहते हैं। Computer में जो भी कुछ होता है वह File के रूप में होता है। Computer में दो तरह की File होती है। पहली वह File होती है जिसमें हम हमारे महत्वपूर्ण Data Store करके रखते हैं। इसे **Data File** कहा जाता है। दूसरी File वह File होती है, जिसमें Computer के लिए वे Instructions होती हैं, जो Computer को बताती हैं कि उसे किसी Data पर किस प्रकार से Processing करके Result Generate करना है। इस दूसरी प्रकार की File को **Program File** कहा जाता है।

हम विभिन्न प्रकार की Computer Languages में Program Files ही Create करते हैं। जब बहुत सारी Program Files मिल कर किसी समस्या का समाधान प्राप्त करवाती हैं, तो उन Program Files के समूह को **Software** कहा जाता है। Computer Software मुख्यतया दो प्रकार के होते हैं:

System Software:

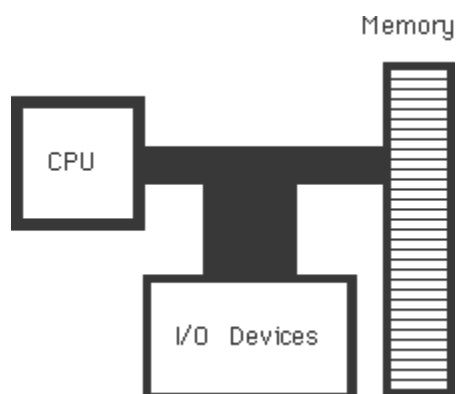
ये Software उन प्रोग्रामों का एक समुह होता है जो कम्प्यूटर की Performance को Control करता है। यानी Computer पर किस तरह से एक प्रोग्राम रन होगा और किस तरह से प्रोग्राम Output देगा। किस तरह Hard Disk पर Files Save होंगी, किस तरह पुनः प्राप्त होंगी, आदि। Windows, Unix, Linux, आदि System Software के उदाहरण हैं।

Application Software:

ये Software प्रोग्रामरों द्वारा लिखे जाते हैं व ये Software किसी खास प्रकार की समस्या का समाधान प्राप्त करने के लिए होते हैं। जैसे Tally, MS-Office आदि Application Software के उदाहरण हैं।

Computer Architecture

Computer से अपना मनचाहा काम करवाने के लिए, सबसे पहले हमें Computer के Architecture को समझना होगा। Computer के Architecture को समझे बिना, हम Computer Programming को ठीक से नहीं समझ सकते। Computer System के मुख्य-मुख्य तीन भाग होते हैं—



I/O Devices

वे Devices जिनसे Computer में Data Input किया जाता है और Computer से Data Output में प्राप्त किया जाता है, I/O Devices कहलाती हैं। Keyboard एक Standard Input Device है और Monitor एक Standard Output Device है।

Center Processing Unit (CPU)

यह एक Microprocessor Chip होता है। इसे Computer का दिमाग भी कहा जाता है क्योंकि Computer में जो भी काम होता है, उन सभी कामों को या तो CPU करता है या Computer के अन्य Devices से उन कामों को करवाता है। इसका मुख्य काम विभिन्न प्रकार के Programs को Execute करना होता है। इस CPU में भी निम्न विभाग होते हैं जो अलग-अलग काम करते हैं:

Control Unit

इस Unit का मुख्य काम सारे Computer को Control करना होता है। CPU का ये भाग Computer की आंतरिक प्रक्रियाओं का संचालन करता है। यह Input/Output क्रियाओं को Control करता है, साथ ही ALU व Memory के बीच Data के आदान-प्रदान को निर्देशित करता है।

यह Program को Execute करने के लिए Program के Instructions को Memory से प्राप्त करता है और इन Instructions को Electrical Signals में Convert करके उचित Devices तक पहुंचाता है, जिससे Data पर Processing हो सके। Control Unit ALU को बताता है कि Processing के लिए Data Memory में कहां पर स्थित हैं, Data पर क्या प्रक्रिया करनी है और Processing के बाद Data को वापस Memory में कहां पर Store करना है।

Arithmetic Logic Unit (ALU)

CPU के इस भाग में सभी प्रकार की अंकगणितीय व तार्किक प्रक्रियाएं होती हैं। इस भाग में ऐसा Electronic Circuit होता है जो Binary Arithmetic की गणनाएं करता है। ALU Control Unit से निर्देश या मार्गदर्शन लेता है, Memory से Data प्राप्त करता है और परिणाम को या Processed Data को वापस Memory में ही Store करता है।

Registers

Microprocessor में कुछ ऐसी Memory होती है जो थोड़े समय के लिए Data को Store कर सकती है। इन्हें Registers कहा जाता है। Control Unit के निर्देशानुसार जो भी Program Instructions व Data Memory से आते हैं वे ALU में Calculation के लिए इन्हीं Registers में Store रहते हैं। ALU में Processing के बाद वापस ये Data Memory में Store हो जाते हैं।

Memory

Memory Computer की Working Storage या कार्यकारी मेमोरी होती है। यह Computer का सबसे महत्वपूर्ण भाग होता है। इसे RAM कहते हैं। इसी में Process होने वाले Data और Data पर Processing करने के Program Instructions होते हैं, जिन्हें Control Unit ALU में Processing के लिए Registers में भेजता है। Processing के बाद जो सूचनाएं या Processed Data Generate होते हैं, वे भी Memory में ही आकर Store होते हैं।

Memory में Data को संग्रह करने के लिए कई Storage Locations होती हैं। हर Storage Location एक Byte की होती है और हर Storage Location का एक पूर्णांक Number होता है जिसे उस Memory Location का Address कहते हैं।

हर Storage Location की पहचान उसके Address से होती है। 1 Byte की RAM में एक ही Character Store हो सकता है और इसमें सिर्फ एक ही Storage Location हो सकती है। इसी तरह 1 KB की RAM में 1024 Storage Locations हो सकती हैं और इसमें 1024 अक्षर Store हो सकते हैं। जो Memory जितने Byte की होती है उसमें उतने ही Characters Store हो सकते हैं और उसमें उतनी ही Storage Locations हो सकती हैं।

जिस तरह से किसी शहर में ढेर सारे घर होते हैं और हर घर का एक Number होता है। किसी भी घर की पहचान उसके घर के Number से भी हो सकती है। उसी तरह से Memory में भी विभिन्न Storage Cell होते हैं जिनका एक Unique Number होता है। हम किसी भी Storage Cell को उसके Number से पहचान सकते हैं और Access कर सकते हैं। हर Storage Cell के इस Unique Number को उस Storage Cell का Address कहते हैं।

जिस तरह से हम किसी घर में कई तरह के सामान रखते हैं और जरूरत होने पर उस घर से उस सामान को प्राप्त करके काम में ले लेते हैं, उसी तरह से Memory में भी अलग-अलग Storage Cells में हम अपनी जरूरत के अनुसार अलग-अलग मान Store कर सकते हैं और जरूरत पड़ने पर उस Data को प्राप्त कर के काम में ले सकते हैं।

Types of Programming

Computer एक Digital Machine है। Computer तभी कोई काम कर सकता है जब उसे किसी काम को करने के लिए Program किया गया हो। Programming दो तरह की होती है:

एक Programming वह होती है जो किसी Computer को काम करने लायक अवस्था में लाने के लिए की जाती है। इस Programming को भी दो भागों में बांटा जा सकता है :

Hardware Programming

इस Programming के अन्तर्गत Computer के Hardware यानी Computer के Motherboard पर लगाए गए विभिन्न प्रकार के Chips व Computer से जुड़े हुए अन्य विभिन्न प्रकार के Peripherals जैसे कि Keyboard, Mouse, Speaker, Monitor, Hard Disk, Floppy Disk, CD Drive आदि को Check करने व Control करने के लिए हर Mother Board पर एक **BIOS Chip** लगाई जाती है। इस BIOS Chip का मुख्य काम Computer को ON करते ही विभिन्न प्रकार के Devices को Check करना होता है। यदि Computer के साथ जुड़ी हुई कोई Device ढंग से काम नहीं कर रही है, तो BIOS User को विभिन्न प्रकार की Error Messages देता है।

BIOS Chip के अन्दर ही प्रोग्राम को लिखने का काम BIOS बनाने वाली Company करती है। इसे Hard Core Programming या Firmware कहा जाता है। Hardware Programming में Chip को बनाते समय ही उसमें Programming कर दी जाती है। किसी भी Computer के Motherboard पर लगी BIOS Chip यदि खराब हो जाए, तो Computer किसी भी हालत में काम करने लायक अवस्था में नहीं आ सकता यानी Computer कभी Boot नहीं होता।

Software Programming

Computer को काम करने लायक अवस्था में लाने के लिए जिस Software को बनाया जाता है, उसे Operating System Software कहा जाता है। BIOS Chip का काम पूरा होने के बाद Computer का पूरा Control Operating System Software के पास आ जाता है। Computer के पास BIOS से Controlling आने के बाद सबसे पहले Memory में Load होने वाला Software Operating System Software ही होता है। इसे **Master Software** भी कहते हैं।

आज विभिन्न प्रकार के Operating System Software बन चुके हैं जैसे DOS, Windows, OS/2, WRAP, Unix, Linux आदि। इन सभी Software का मुख्य काम Computer को Boot करके User के काम करने योग्य अवस्था में लाना होता है।

दूसरी Programming वह Programming होती है, जिससे Computer हमारी बात को समझता है और हमारी इच्छानुसार काम करके हमें परिणाम प्रदान करता है। इन्हें **Application Software** कहा जाता है।

हम किसी भी Operating System के लिए किसी भी भाषा में जब कोई Program लिखते हैं, तो वास्तव में हम Application Software ही लिख रहे होते हैं। Application Software का मुख्य काम किसी विशेष समस्या का समाधान प्रदान करना होता है। MS-Office, Corel-Draw, PageMaker, Photoshop आदि Application Software के उदाहरण हैं, जो हमें किसी विशेष समस्या का समाधान प्रदान करते हैं। जैसे यदि हमें Photo Editing से सम्बंधित कोई काम करना हो, तो हम Photoshop जैसे किसी Application Software को उपयोग में लेते हैं।

LANGUAGE INTRODUCTION

Language Introduction

भाषा, दो व्यक्तियों के बीच संवाद, भावनाओं या विचारों के आदान-प्रदान का माध्यम प्रदान करती है। हम लोगों तक अपने विचार पहुंचा सकें व अन्य लोगों के विचारों का लाभ प्राप्त कर सकें इसके लिए जरूरी है कि संवाद स्थापित करने वाले दोनों व्यक्तियों के बीच संवाद का माध्यम समान हो। यही संवाद का माध्यम भाषा कहलाती है। अलग-अलग स्थान, राज्य, देश, परिस्थितियों के अनुसार भाषा भी बदलती रहती है, लेकिन सभी भाषाओं का मकसद संदेशों या सूचनाओं का आदान प्रदान करना ही होता है।

ठीक इसी तरह कम्प्यूटर की भी अपनी कई भाषाएं हैं, जो जरूरत व उपयोग के अनुसार विकसित की गई हैं। हम जानते हैं, कि कम्प्यूटर एक इलेक्ट्रॉनिक मशीन मात्र है। ये हम सजीवों की तरह सोच विचार नहीं कर सकता है और ना ही हमारी तरह इनकी अपनी कोई भाषा है, जिससे हम इनसे सम्बंध बना कर सूचनाओं का लेन-देन कर सकें। इसलिए कम्प्यूटर को उपयोग में लेने के लिए एक ऐसी भाषा की जरूरत होती है, जिससे हम हमारी भाषा में कम्प्यूटर को सूचनाएं दें व कम्प्यूटर उसे उसकी मशीनी भाषा में समझे और हमारी चाही गई सूचना या परिणाम को हमें हमारी भाषा में दे ताकि हम उसे हमारी भाषा में समझ सकें।

Level of Computer Languages

कम्प्यूटर मुख्यतः एक ही भाषा यानी मशीनी भाषा को ही समझता है। फिर भी मोटे तौर पर कम्प्यूटर भाषा को निम्नानुसार तीन भागों में बांटा गया है। ये High Level Languages हैं, जिनमें एक ऐसा Software या Program होता है जो इन High Level Languages के Program Codes को मशीनी भाषा के Low Level Codes में Convert करने का काम करता है, जिन्हें Computer समझता है।

Low Level Language or Machine Language

इसे मशीनी भाषा भी कहते हैं। यह भाषा केवल बाइनरी कोड के अनुसार लिखनी होती है, इसलिए ये भाषा केवल वे ही लोग उपयोग में ले सकते हैं जो कम्प्यूटर की सारी आंतरिक संरचना को जानते हों साथ ही इस भाषा में लिखे प्रोग्राम केवल उसी कम्प्यूटर पर चलते हैं, जिस पर ये लिखे जाते हैं। यह एक बहुत ही कठिन भाषा होती है।

Middle Level or Assembly Language

इसे असेम्बली भाषा भी कहते हैं। इस भाषा में सामान्य अंग्रेजी के शब्दों को उपयोग में लेकर प्रोग्राम लिखा जाता है इसलिए ये भाषा उपयोग में मशीनी भाषा से सरल होती है, लेकिन फिर भी काफी जटिल होती है। इसमें एक असेम्बलर होता है, जो सामान्य अंग्रेजी के शब्दों को मशीनी भाषा में बदलने का काम करता है ताकि कम्प्यूटर उसे समझ सके। इस भाषा में भी प्रोग्राम बनाने वाले प्रोग्रामर को कम्प्यूटर हार्डवेयर का सम्पूर्ण ज्ञान होना जरूरी होता है व ये प्रोग्राम भी उसी कम्प्यूटर पर Run हैं, जिस पर इन्हे लिखा गया हो।

High Level Language

ये हमारे आज के वातावरण में उपयोग में आने वाली भाषाएं हैं। ये भाषाएं इतनी सरल हैं कि कोई भी सामान्य व्यक्ति इनमें प्रोग्राम बना सकता है। इसमें सारे के सारे कोड अंग्रेजी में लिखे जाते हैं व इसमें एक कम्पायलर होता है जो सीधे ही प्रोग्राम को मशीनी कोड में बदल देता है।

Assembler

Assembly Language में लिखे प्रोग्राम को मशीनी भाषा में बदलने का काम **Assembler** करता है। ये एक ऐसा Software होता है, जो किसी Text File में लिखे गए विभिन्न Assembly Codes को Computer की मशीनी भाषा में Convert करके Computer के CPU पर Process करता है। Computer का CPU उन Converted Codes को समझता है और हमें हमारा वांछित परिणाम उस भाषा में प्रदान करता है, जिस भाषा को हम समझ सकते हैं यानी CPU हमें सामान्य English भाषा में Processed Results प्रदान करता है।

Compiler and Interpreter

Compiler व Interpreter भी High Level Program Codes को मशीनी भाषा में बदलने का काम करते हैं लेकिन दोनों के काम करने के तरीके में कुछ अन्तर हैं। Compiler पूरे प्रोग्राम को एक ही बार में मशीनी भाषा में बदल देता है व सभी Errors को Debug करने के बाद एक Executable Program File Provide करता है, जो कि एक Machine Language Code File होती है। इस Machine Language Code File को फिर से Compile करने की जरूरत नहीं होती है। जबकि Interpreter प्रोग्राम की हर लाइन को हर बार मशीनी कोड में बदलता है, जिससे एक Interpreted Program को हर बार Run करने के लिए Interpret करना जरूरी होता है। HTML Code File Interpreted Program का एक उदाहरण है, जिसे हर बार Run होने के लिए Web browser Interpreter की जरूरत होती है।

Similarities between Real Word and Computer Program

प्रोग्राम को हम हर रोज के हमारे दैनिक जीवन के कामों से भी समझ सकते हैं। जिस तरह हमें कोई सामान्य सा काम के लिए भी एक निश्चित क्रम का पालन करना पड़ता है, उसी तरह कम्प्यूटर को भी एक निश्चित क्रम में सूचनाएं देनी होती हैं, कि किस काम के बाद क्या काम करना है। ताकि एक निश्चित समाधान या मनचाहा परिणाम प्राप्त किया जा सके। उदाहरण के लिए, माना हमें कुछ सामान खरीदने के लिए बाजार जाना है, तो हमें निम्न क्रम में अपना काम करना पड़ेगा :

- किस समय बाजार जाए ताकि अधिकतर दुकाने खुली हों और भीड़ कम हो ?
- किस दिन सस्ता सामान मिल सकेगा ?
- क्या-क्या खरीदना है ?
- कितने रूपयों की जरूरत होगी ?
- किस सवारी से जाना है ?
- किसके साथ बाजार जाना है ?
- खरीददारी के साथ और क्या काम किया जा सकता है ? आदि – आदि

ठीक इसी तरह से "सी" Language में भी प्रोग्राम बनाया जाता है। यानी कामों का एक सुव्यवस्थित समूह Create किया जाता है और उस समूह को Computer के समझने योग्य Programming Language में Coding के रूप में एक File में लिख दिया जाता है। इस File को Program की **Source File** कहते हैं।

जिस File में Computer के समझने योग्य Coding के रूप में विभिन्न Steps या Instructions को लिखे गए होते हैं, उस File को Compile किया जाता है। Source File को Compile करने पर एक नई File बनती है, जिसके Instructions को Computer का CPU समझ सकता है। इस Compiled

File को **Executable File** या **Exe File** कहा जाता है, क्योंकि Compiling के बाद Create होने वाली इस नई File का Extension **.EXE** होता है।

अब हमें जब भी वह काम करना होता है, जिसके लिए हमने Program लिखा है, तो हमें Source File को वापस से Compile करने की जरूरत नहीं होती है। हमें केवल उस Create होने वाली नई Executable File को ही Run करना होता है। इस File में CPU को जो कुछ करना है उसकी Instructions होती हैं जिन्हे CPU समझ सकता है। इस प्रकार से Computer में एक Program Create होता है।

इस पूरे Discussion के आधार पर यदि हम किसी Computer Program की परिभाषा देना चाहें तो ये कह सकते हैं कि **Computer Instructions** का एक ऐसा सुव्यवस्थित क्रम, जिससे Computer द्वारा किसी समस्या का उचित समाधान प्राप्त हो सके, **Program** कहलाता है।

Steps of Program

- 1 (Problem Definition)** प्राग्राम परिभाषण
इस चरण में उस समस्या को पूरी तरह से समझना होता है, जिसका प्रोग्राम बना कर कम्प्यूटर से समाधान प्राप्त करना है। यानी प्रोग्राम के द्वारा हमें क्या प्राप्त परिणाम करना है, यह निष्कर्ष निकालना होता है।
सारांश :- क्या परिणाम प्राप्त करना है ?
- 2 (Problem Design)** प्रोग्राम डिजाइन
इस चरण में समस्या को कई भागों में बांट कर उसे बीजगणितीय एल्गोरिदम के अनुसार लिख लिया जाता है। एल्गोरिदम लिखने के लिए फ्लोचार्ट आदि को उपयोग में लिया जाता है।
सारांश :- कैसा परिणाम प्राप्त करना है ?
- 3 (Program Coding)** कोडिंग
इस चरण में हाई लेवल भाषा के कोडों के अनुसार एल्गोरिदम व फ्लोचार्ट की मदद से प्रोग्राम की कोडिंग की जाती है।
सारांश :- कब क्या होगा जब User इसे उपयोग में लेगा ?
- 4 (Program Execution)** प्रोग्राम को **Execute** करना
इस चरण में बनाए गए प्रोग्राम को चलाया जाता है।
- 5 (Program Debugging)** डीबगिंग
जब प्रोग्राम को बनाया जाता है, तब कई तरह की गलतियां रह जाती हैं। जिससे जब प्रोग्राम को चलाया जाता है तब या तो प्रोग्राम रन नहीं होता या फिर सही परिणाम प्राप्त नहीं होता है। जब प्रोग्राम को कम्पाइल किया जाता है तो कम्पायलर में एक डीबगर होता है, जो प्रोग्राम में जिस जगह पर गलती होती है, वहीं पर आकर रुक जाता है। हम वहां पर होने वाली बग को सही करके प्रोग्राम को पुनः रन करते हैं। प्रोग्राम में होने वाली गलतियों को ढूंढना व उन्हें सही करना ही डीबगिंग कहलाता है।
सारांश :- प्रोग्राम की किसी भी तरह की व्याकरण सम्बंधी या तर्क सम्बंधी गलती को खोजना व उसे संसोधित करके प्रोग्राम को सही करना।
- 6 (Program Testing)** प्रोग्राम टेस्टिंग
कई बार प्रोग्राम पूरी तरह सही रन होता है, लेकिन फिर भी उसमें गलती होती है। इसे तार्किक गलती कहते हैं। इस प्रकार की गलती से हमें वांछित सही परिणाम प्राप्त नहीं होता है। इसे सुधारने के लिए

प्रोग्राम से ऐसी समस्याओं का हल मांगा जाता है, जिसका परिणाम हमें पहले से ही पता होता है। ऐसा करने से यदि प्रोग्राम में कहीं पर तार्किक कमी हो तो पता चल जाता है। इस प्रक्रिया को प्रोग्राम टेस्टिंग करना कहते हैं।

7 (Program Documentation) प्रोग्राम विवरण

कई बार प्रोग्राम इतने बड़े व जटिल हो जाते हैं कि कब कहां और क्या होना है और कौनसा प्रोग्राम क्यों लिखा गया था इसका पता ही नहीं चल पाता है। इस तरह की समस्याओं से बचने के लिए प्रोग्राम में कई जगहों पर ऐसी टिप्पणीयां डाल दी जाती हैं, जिससे पता चल सके कि प्रोग्राम क्या है व वह प्रोग्राम किसलिए लिखा गया है।

Characteristics of a Good Program

प्रोग्राम लिखते समय हमें कई बिंदुओं को ध्यान में रखना होता है। इसमें से कुछ खास बिन्दु निम्नानुसार हैं:

1 (Reliability) विश्वसनीयता

यह जरूरी है कि प्रोग्राम बिना किसी व्यवधान के वही काम करे जिसके लिए उसे बनाया गया है। माना कि हमने एक ऐसा प्रोग्राम बनाया जिसमें किसी भिन्नात्मक संख्या का हर कोई वेरियेबल है, जो घटते-घटते अन्त में शून्य हो जाता है। ऐसी दशा में संख्या का भागफल अनन्त हो जाएगा क्योंकि किसी भी संख्या में शून्य का भाग देने पर भागफल अनन्त प्राप्त होता है, जिससे प्रोग्राम सही परिणाम नहीं देगा। इस प्रकार की गलतियों का ध्यान रखना चाहिये।

2 (Flexibility) लचीलापन

प्रोग्राम इस तरह का होना चाहिये कि जब भी भविष्य में कभी जरूरत पड़े, तो उसमें नया कुछ जोड़ा जा सके या अनावश्यक चीजों को हटाया जा सके। इसे प्रोग्राम की **Maintainability** कहा जाता है। जैसे कि किसी प्रोग्राम में 20 वर्षों का ब्याज निकालने की व्यवस्था है, तो उसमें यह ऐसी सुविधा होनी चाहिये कि आवश्यकता होने पर कुछ फेर बदल करके 25 वर्षों का ब्याज भी निकाला जा सके।

3 (Portability)

प्रोग्राम इस तरह लिखा होना चाहिये कि एक Computer पर Develop किया गया Program बिना फिर से Compile किए हुए किसी दूसरे Computer पर भी आसानी से Execute हो सके।

4 (Readability) सुपाठ्यता

प्रोग्राम में जगह-जगह पर कई ऐसी टिप्पणीयां होनी चाहिये जिससे प्रोग्राम का **Flow** व प्रोग्राम का उद्देश्य पता चलता रहे।

5 (Performance)

प्रोग्राम द्वारा कम से कम समय में अच्छा से अच्छा परिणाम प्राप्त होना चाहिये।

Problem – Doing Something

Computer द्वारा हम किसी ना किसी प्रकार की समस्या का समाधान प्राप्त करने के लिए ही विभिन्न प्रकार के Programs लिखते हैं। इसलिए सबसे पहले हमें यही तय करना होगा कि आखिर हम Computer के संदर्भ में किस बात को एक समस्या के रूप में देख सकते हैं ?

यदि बिल्कुल ही सरल शब्दों में किसी समस्या को परिभाषित करें, तो **Computer** पर हम जिस किसी भी काम को **Perform** करके किसी प्रकार का कोई **Result** प्राप्त करना चाहते हैं, हम उस काम को समस्या के रूप में देख सकते हैं। उदाहरण के लिए दो संख्याओं का योग **करना**, किसी परिणाम को **Computer** के **Monitor** पर **Display करना**, किसी भी प्रकार की कोई **Calculation** या **Comparison करना** आदि इन सभी कामों को हम समस्या के रूप में देख सकते हैं। यानी हम जो कुछ भी **करना** चाहते हैं, वह सबकुछ **Computer** के लिए एक समस्या ही है।

Algorithm – List of Sequential Steps to Solve a Problem

हम हमारे दैनिक जीवन में जिस किसी भी काम को भी करते हैं, उस काम को **Problem** कह सकते हैं। हर **Problem** को **Solve** करने का एक निश्चित क्रम होता है और इस निश्चित क्रम के अन्तर्गत हमें विभिन्न प्रकार के **Steps Use** करने होते हैं। उदाहरण के लिए मानलो कि हमें किसी को **Phone करना** है। ये भी एक तरह की समस्या ही है क्योंकि हमें कुछ **करना** है। अब **Phone** करने के लिए हमें निम्न काम करने होते हैं:

- 1 सबसे पहले हम **Phone** को इस बात के लिए **Check** करेंगे, कि **Phone** चालू है या नहीं। यानी **Dial Tone** आ रही है या नहीं।
- 2 यदि **Dial Tone** आ रही है, तो हमें उस व्यक्ति का **Phone Number Dial** करना होता है, जिससे हम बात करना चाहते हैं।
- 3 **Phone Number Dial** करने के बाद हमें **Target** व्यक्ति के **Phone** पर **Bell** जाने का इन्तजार करना होगा। यदि **Bell** जाती है, तो **Target** व्यक्ति **Phone** उठाएगा और बात हो जाएगी।

इन **Steps** के समूह से हम समझ सकते हैं कि हमें **Phone** करने जैसी मामूली सी समस्या को सुलझाने के लिए भी एक निश्चित क्रम का पालन करना जरूरी होता है, साथ ही सभी जरूरी **Steps Follow** करने भी जरूरी होते हैं। ना ही हम इन **Steps** के क्रम को **Change** कर सकते हैं और ना ही हम किसी **Step** को छोड़ सकते हैं। यदि हम इन दोनों में से किसी भी एक बात को **Neglect** करते हैं, तो हम **Target** व्यक्ति से बात नहीं कर सकते हैं, यानी समस्या का **Solution** प्राप्त नहीं कर सकते हैं।

इस उदाहरण का सारांश ये है कि किसी भी समस्या का एक निश्चित व उचित समाधान प्राप्त करने के लिए हमें उस समस्या को विभिन्न प्रकार के **Steps** के एक समूह के रूप में **Define** करना होता है, जो कि एक निश्चित क्रम में होते हैं। **Steps** के इस समूह को ही **Algorithm** कहा जाता है।

दूसरे शब्दों में कहें तो किसी भी समस्या के एक निश्चित समाधान को प्राप्त करने के लिए अनुक्रमिक व चरणबद्ध रूप में अपनाई जाने वाली लिखित प्रक्रिया को हम **एल्गोरिद्म** कहते हैं।

उदाहरण के लिए मानलो कि हम दो संख्याओं **A** व **B** को जोड़ कर उसका परिणाम **C** में प्राप्त **करना** चाहते हैं और फिर **C** के मान को **Monitor** पर **Display** करना चाहते हैं। यानी हमें **C = A + B** करना है। इस काम को पूरा करने के लिए या इस समस्या को सुलझाने के लिए हमें निम्नानुसार क्रम का पालन करना होता है:

हल :- चरण 1 प्रक्रिया का प्रारम्भ।

चरण 2 वेरिएबल **A** का मान पढना।

चरण 3 वेरिएबल **B** का मान पढना।

चरण 4 **A** व **B** के मान का योग निकालना।

चरण 5 मान **A** व **B** के योगफल को **Variable C** के स्थान पर रखना।

चरण 6 **C** के मान को प्रिंट करना।

चरण 7 प्रक्रिया का अंत करना।

History of Programming Language “C”

इस भाषा का विकास होने से पहले जितने भी Program बनाए जाते थे, वे सभी Assembly Language में बनाए जाते थे। Assembly Language में बनाए गए Programs की Speed काफी ज्यादा होती है, लेकिन इसकी एक कमी भी है। Assembly Language में Develop किया गया Program उसी Computer पर Execute होता है, जिस पर उसे Develop किया गया होता है।

इसलिए एक ऐसी Programming Language की आवश्यकता हुई, जो कि Portable हो। इस जरूरत के आधार पर सन् 1960 में केम्ब्रिज यूनिवर्सिटी ने एक कम्प्यूटर प्रोग्रामिंग भाषा का विकास किया, जिसका नाम “BASIC COMBINED PROGRAMMING LANGUAGE” यानी **BCPL** रखा गया। सन् 1970 में केन थॉम्पसन ने इसमें कुछ परिवर्तन किये व सामान्य बोलचाल में इसे “B” भाषा कहा। “C” का विकास अमेरिका में सन् 1972 में हुआ। AT & T Laboratory के कम्प्यूटर वैज्ञानिक डेनिस रिची ने इस का विकास किया था।

“सी” एक शक्तिशाली भाषा है जिसमें हम एप्लीकेशन सॉफ्टवेयर व सिस्टम सॉफ्टवेयर दोनों तरह के सॉफ्टवेयर बना सकते हैं। इसमें सामान्य अंग्रेजी शब्दों के माध्यम से प्रोग्राम बनाए जाते हैं, जो कि समझने व बनाने में आसान होते हैं। “सी” एक हाई लेवल Structured Programming Language भाषा है, यानी सूचनाओं के एक निश्चित क्रम में Program Run होता है।

Characteristics of “C”

“सी” अन्य कई भाषाओं से काफी सरल है। अन्य हाई लेवल भाषाओं की तुलना में “सी” काफी लचीली भाषा है। “सी” ही एक ऐसी भाषा है, जिसमें कम्प्यूटर के हार्ड वेयर के साथ भी काम किया जा सकता है। इसके द्वारा मेमोरी मेनेजमेन्ट किया जा सकता है। सबसे बड़ी खासियत “सी” की पोर्टेबिलिटी है।

यानी “सी” भाषा में लिखे गए प्रोग्राम किसी भी अन्य कम्प्यूटर वातावरण में चल सकते हैं। “सी” एक फंक्शनल भाषा है यानी इसमें सभी काम विभिन्न प्रकार के फंक्शनस् को यूज करके किया जाता है। “सी” में कोई इनपुट आउटपुट ऑपरेशन नहीं है। “सी” कम्पाइलर सभी इनपुट आउटपुट का काम लाइब्रेरी फंक्शन के द्वारा करता है।

Block Structure of “C” Programs

Documentation Section
Link Section
Definition Section
Global Declaration Section
Main() Function Section <pre>{ Declaration Part Executable Part }</pre>
Sub Program Section Function 1 Function 2 ... Function n

Layout Structure of “C” Programs

```

1      /* Comment about the Program */
2      Including The Header Files
3      Global Variables Declaration
4      Main()
5      {
6          Local Variables Declaration
7          Necessary Statements
8      }
9      Sub Program Functions
      Function 1
      Function 2
      ;
      Function n

```

1 Documentation Section

प्रोग्राम के इस भाग में हम प्रोग्राम से सम्बन्धित कुछ बिन्दु टिप्पणी के रूप में लिखते हैं, ताकि प्रोग्राम किस कारण से बनाया गया है और प्रोग्राम की विशेषता क्या है, ये बताया जा सके।

2 Link Section

यहां पर हम “सी” प्रोग्राम की उन हेडर फाइलों को डिक्लेयर करते हैं, जिनकी हमारे प्रोग्राम में आवश्यकता है।

3 Definition Section

यहां उन वेरियेबल्स को डिफाईन किया जाता है जिनका प्रोग्राम में सीधे ही उपयोग हो सकता हो। ये एक तरह से स्थिरांक होता है। इसे ग्लोबल कॉन्स्टेंट भी कह सकते हैं।

4 Global Declaration Section

जिस किसी वेरियेबल को इस स्थान पर डिक्लेयर कर दिया जाता है, उस वेरियेबल को प्रोग्राम में कहीं भी उपयोग में लिया जा सकता है।

5 Main() Function Section

यह फंक्शन हर "सी" प्रोग्राम में होता है। कम्पाइल करते समय Program Control हमेशा main() Function को ही ढूँढता है। हर "सी" प्रोग्राम में सिर्फ एक ही main() Function हो सकता है व हर "सी" प्रोग्राम में main() Function का होना जरूरी होता है क्योंकि Program का Execution हमेशा main() Function से ही शुरू होता है।

6 { Opening Parenthesis

main() Function मिलने के बाद प्रोग्राम का एकजीक्यूशन इसी मंज़ले कोष्ठक से शुरू होता है।

7 Declaration Part

प्रोग्राम में काम आने वाले सभी वेरियेबल्स, कॉन्स्टेंट, एरे आदि को यहीं पर डिक्लेयर करना होता है। यहां पर हम जिसे भी डिक्लेयर करते हैं, उसके लिए "सी" प्रोग्राम Execution के समय मेमोरी में जगह बना देता है, जिन्हें बाद में अपनी आवश्यकता के अनुसार उपयोग में लिया जाता है।

8 Executable Part

यहां पर प्रोग्राम के वे सभी स्टेटमेंट्स होते हैं जिनके द्वारा हम प्रोग्राम से कोई परिणाम प्राप्त करना चाहते हैं। यही वह भाग होता है जहां से User के लिए Interface का काम शुरू होता है।

9 } Closing Parenthesis

प्रोग्राम में दूसरे मंज़ले कोष्ठक का प्रयोग वहां करते हैं, जहां पर प्रोग्राम का अन्त करना होता है।

Sub Program Section

```
Function 1;  
Function 2;  
...  
...  
Function n;
```

प्रोग्राम के इस भाग में यूजर डिफाइन फंक्शन होते हैं। एक main() प्रोग्राम में main() Function तो एक ही होता है लेकिन User Defined Function आवश्यकता के अनुसार कई हो सकते हैं।

Coding Structure of "C" Programs

सबसे पहले किसी प्रोग्राम की कोडिंग की जाती है। फिर प्रोग्राम को कम्पाइल किया जाता है। कम्पाइल करने से प्रोग्राम की हाई लेवल के कोड मशीनी भाषा के बाइनरी डिजिट्स में बदल जाते हैं, जिन्हें हमारा Computer समझ सकता है। हम "सी" प्रोग्राम के एकजीक्यूशन को एक ब्लॉक डायग्राम या Flow Chart से समझाने की कोशिश कर रहे हैं।

सबसे पहले कम्प्यूटर चालू करेंगे और "सी" भाषा के कोडों को लिख कर प्रोग्राम बनाएंगे। इसे **Source Program** कहते हैं। प्रोग्राम बनाने के बाद इसकी किसी भी प्रकार की व्याकरण सम्बंधी गलती को Edit Source Program Block में Edit करके सही करते हैं।

अब "सी" कम्पाइलर द्वारा प्रोग्राम को कम्पाइल करते हैं, जिससे प्रोग्राम को कम्प्यूटर अपनी मशीनी भाषा में समझ सके। यदि इस प्रोग्राम में कोई अन्य वाक्य रचना सम्बंधी गलती हो, तो प्रोग्राम कंट्रोल

पुनः सभी गलतियों के साथ Source Editing के लिए उसी Edit Source Program Block में चला जाता है।

जब प्रोग्राम में किसी भी प्रकार की कोई व्याकरण सम्बंधी गलती नहीं रह जाती है, तब Program Control उन **System Library Files** को प्रोग्राम में लिंक करता है, जिनके Function Program में Use हुए हैं।

जैसे Input/Output के सारे Functions **stdio.h** नाम की Header File में Store रहते हैं, इसलिए I/O की सुविधा प्राप्त करने के लिए इस Header File को हर C Program में Include किया जाता है।

जब Program Control सभी आवश्यक Header Files को Program से Link कर देता है। फिर अगली Stage में यूजर से Data Input करवाया जाता है व प्रोग्राम Execute होता रहता है। अब यदि किसी प्रकार की तार्किक गलती हो तो वह गलती अगले प्रोसेस बॉक्स में पकड में आती है।

यदि गलती है, तो प्रोग्राम Control पुनः Edit Source Program Block में पहुंच जाता है, और सारी की सारी प्रक्रिया पुनः प्रोग्राम को डिबग करने में अपनाई जाती है। लेकिन यदि प्रोग्राम में कोई Error नहीं हो तो प्रोग्राम Correct Output देता है और समाप्त हो जाता है। इस तरह पूरा प्रोग्राम Step-By-Step Execute होता है।

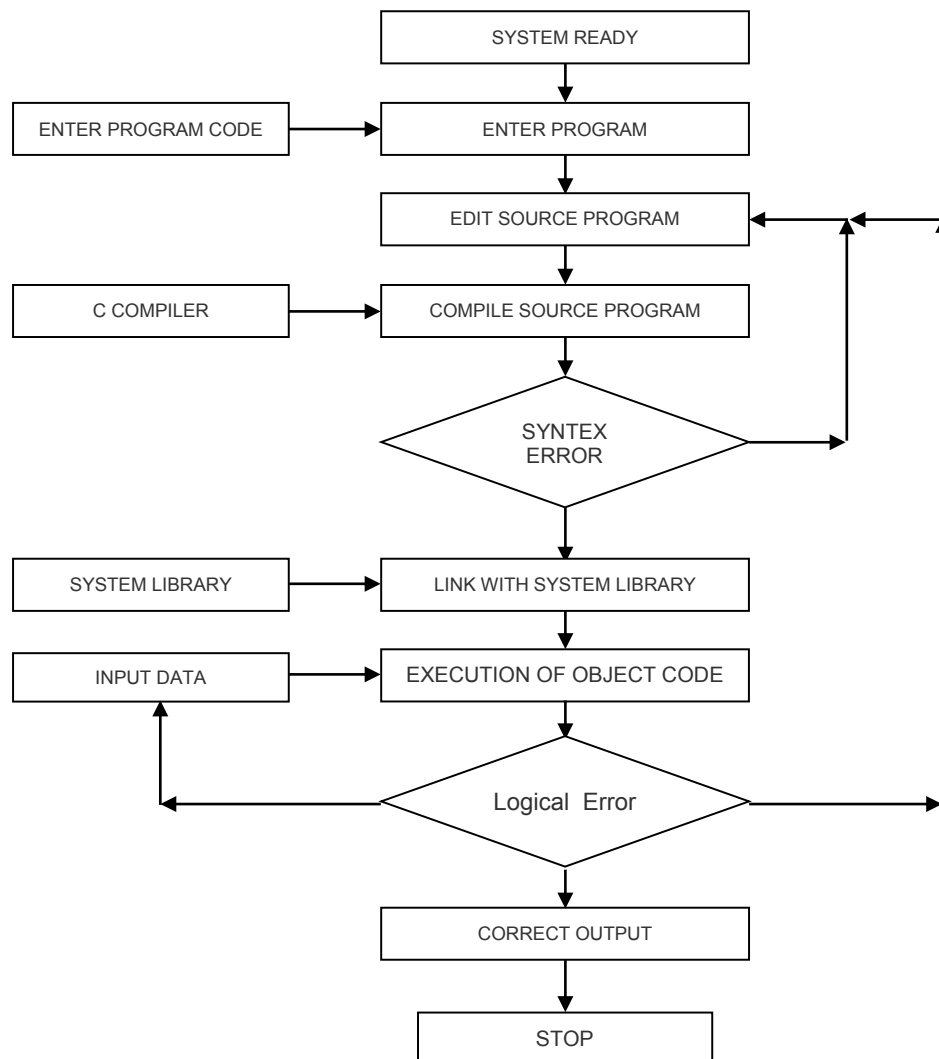
main() Function

```
{  
    Function Body ;  
}
```

यह किसी भी प्रोग्राम का एक अनिवार्य हिस्सा है। जब भी कोई प्रोग्राम कम्पाइल करते हैं तो कम्पाइलर सर्वप्रथम main() Function को ढूँढता है और इसके मंजले कोष्ठक से प्रोग्राम का Execution शुरू करता है। सभी Executables Code इन्हीं मंजले कोष्ठकों के बीच लिखे जाते हैं।

किसी भी Function की शुरुआत व अन्त के Statements इन्हीं मंजले कोष्ठकों के बीच लिखे जाते हैं, फिर चाहे ये User Defined Functions हों या main() Function, Program के हर Statement का अन्त " ; " सेमीकॉलन के चिन्ह द्वारा ही होता है।

Program Flow



Functions – Pre-Defined and User-Defined

“C” भाषा एक **Functional Programming Language** है। जब हम इस भाषा का प्रयोग करके किसी समस्या का समाधान प्राप्त करना चाहते हैं, तब हमें उस समस्या को छोटे-छोटे हिस्सों में बांटना होता है और उन सभी हिस्सों को अलग-अलग Solve करके अन्त में सभी हिस्सों को जोड़ना होता है।

किसी समस्या से सम्बंधित इन विभिन्न प्रकार के छोटे-छोटे हिस्सों को **Function** कहा जाता है। ये Function किसी एक काम को पूरी तरह से पूरा करते हैं और केवल एक ही काम को पूरा करते हैं। यानी हर Function अपने आप में केवल एक ही काम परिपूर्ण तरीके से पूरा करता है।

उदाहरण के लिए जो Function Keyboard से Input लेने का काम करता है, वह Function केवल Keyboard से Input लेने का ही काम करेगा और जो Function किसी Data को Monitor पर Display करने के लिए लिखा गया है, वह Function Data को केवल Screen पर Display करने का काम ही करेगा।

“C” Language में दो तरह के Functions होते हैं:

- 1 जो Functions हमें Directly Use करने के लिए पहले से ही प्राप्त होते हैं, उन्हें **Pre-Defined** या **Built-In Functions** कहा जाता है। उदाहरण के लिए `printf()`, `clrscr()`, `getch()` आदि Functions हमें पहले से ही प्राप्त हैं। इन्हें Use करने के लिए हमें केवल उन Header Files को अपने Source Program में Include करना होता है, जिनमें इन Functions को Define किया गया होता है। जब हम किसी Predefined Function को अपने Source Program में Use करते हैं, तो इस प्रक्रिया को Function Call करना भी कहा जाता है।
- 2 दूसरे प्रकार के Functions वे Functions होते हैं, जिन्हें Programmer अपनी जरूरत के आधार पर Develop करता है। जिन Functions को एक Programmer स्वयं Create करके Use करता है, उन Functions को **User-Defined Functions** कहते हैं। User-Defined Functions बनाना एक Programmer की इच्छा पर निर्भर करता है।

यदि Programmer चाहे, तो वह सभी प्रकार के कामों को बिना किसी प्रकार का User-Defined Function Create किए हुए भी पूरा कर सकता है। लेकिन Functions Create करने से Program की जटिलता में कमी आ जाती है और Program को Debug करना सरल होता है।

चूंकि `main()` Function भी एक Programmer किसी समस्या का समाधान प्राप्त करने के लिए बनाता है, इसलिए `main()` Program को भी User-Defined Function ही कहा जाता है। लेकिन ये एक ऐसा Function होता है, जिसे बनाना जरूरी होता है। यही वह Function होता है, जहां से Compiler Program को Execute करना शुरू करता है।

#include<Header File>

“सी” भाषा में विभिन्न प्रकार के कामों को पूरा करने के लिए फंक्शनों की अपनी एक पूरी लाइब्रेरी है, जिसमें ढेर सारे **Built-In Functions** हैं। विभिन्न प्रकार के Functions को उनके काम करने की प्रकृति के आधार पर विभिन्न प्रकार की Files में Define या परिभाषित किया गया है। Functions की इन Files को “C” भाषा में **Header File** कहा जाता है।

हम जिस किसी भी Function को Use करना चाहते हैं, हमें उससे सम्बंधित Header File को **#include** शब्द के साथ प्रोग्राम में जोड़ना पड़ता है। जैसे Input/Output से सम्बंधित सारे Functions **stdio.h** नाम की Header File में होते हैं। अतः हमें अपने हर सी प्रोग्राम में इस Header File को **#include<stdio.h>** Code द्वारा Link करना जरूरी होता है। यदि हम ऐसा नहीं करते हैं, तो हमें Input व Output की सुविधा प्राप्त नहीं होती है।

यानी इस Header File को अपने Program में Include किए बिना हम हमारे Program में Keyboard से Input नहीं ले सकते हैं Monitor पर Output को Display नहीं कर सकते हैं। इसी तरह से हमें आउटपुट स्क्रीन पर दिखाई दे रहे पिछले Program के विभिन्न Statements को साफ करके Screen को Clear करना है, तो **clrscr()** Function को Use करना होता है, जो कि **conio.h** नाम की Header File में Defined है, अतः हमें हमारे प्रोग्राम में इस Header File को **#include<conio.h>** Code द्वारा Link करना पड़ेगा।

Header Files को Header File इसलिए कहा जाता है, क्योंकि ये Files किसी भी Source File के Head में यानी सबसे Top पर व सबसे पहले Include की जाती हैं। किसी भी Header File को प्रोग्राम में जोड़ने के लिए **#** के साथ **include** Keyword लगाया जाता है। फिर **< >** के चिन्हों के बीच में उस Header File का नाम लिखा जाता है, जिसे प्रोग्राम में जोड़ना होता है। इनको Declare करने का Syntax निम्नानुसार होता है—

Syntax :	#include <header file name.h>
जैसे :-	#include <stdio.h>
	#include <conio.h>

#define

ये एक **Macro Define** करने का काम करता है। इसका उपयोग **Constant Global Variables Define** करने में किया जाता है। लेकिन इसका उपयोग इतना ही नहीं है। आगे इसके कई उपयोग बताए जाएंगे जो प्रोग्राम Development में काफी मदद करते हैं व प्रोग्राम को अधिक विश्वसनीय व व्यावहारिक बनाने में मददगार होते हैं।

Syntax :	#define Constant Name Constant Value
जैसे :	#define pi 3.142857

ध्यान दें कि स्थिरांक के नाम व उसके मान के बीच किसी प्रकार का कोई चिन्ह नहीं होता है।

जब हम Computer में कोई Program बना कर उस Program के आधार पर किसी समस्या का कोई समाधान प्राप्त करना चाहते हैं, तब हम देखते हैं कि हर Computer Program के हमेशा तीन हिस्से होते हैं, जिन्हें **Input, Process** व **Output** कहा जाता है।

Input Section

Program के Input Section में Program को Use करने वाला User समस्या से सम्बंधित विभिन्न प्रकार के Row Data Input करता है। इन Row Data के आधार पर ही Program अपना आगे का काम सम्पन्न करके कोई Meaningful Result प्रदान करता है। इस Section में User द्वारा Input किए गए विभिन्न प्रकार के मानों को Computer की Memory में Store करने के लिए सभी Data को Memory Allot किया जाता है। User जो भी Data Input करता है, वे सभी Data उनसे सम्बंधित Memory Block में Store हो जाते हैं।

उदाहरण के लिए यदि दो संख्याओं को जोड़ने का Program हो, तो इस Section में कुल तीन Memory Block Allot किए जाते हैं। दो Memory Block दो संख्याओं को Store करने के लिए होते हैं और तीसरा Memory Block उन संख्याओं को जोड़ने से प्राप्त होने वाले परिणाम को Store करने के लिए होता है।

Process Section

इस Section में समस्या से सम्बंधित Input किए गए विभिन्न प्रकार के Data पर विभिन्न प्रकार के Operations Perform करके उचित Result Generate किया जाता है। उदाहरण लिए यदि दो संख्याओं को जोड़ने का Program हो, तो दोनों संख्याओं को जोड़ने का काम इस Section में ही किया जाता है।

Output Section

समस्या से सम्बंधित Input किए गए Data पर Required Operations Perform करने के बाद जो Results Generate होते हैं, उन Results को Monitor पर Display करने या Printer पर Print करने का काम इस Section में किया जाता है।

उदाहरण के लिए दो संख्याओं को जोड़ने पर जो परिणाम प्राप्त होता है, उस परिणाम को इसी Section में Output Devices पर भेजा जाता है। एक User को हमेशा Input व Output Section ही दिखाई देता है, इसलिए Input व Output Section को हमेशा काफी सरल व अच्छे तरीके से Represent करना जरूरी होता है, ताकि User Program से अपनी समस्या का समाधान सरल तरीके से प्राप्त कर सके।

Output Function

“C” Language में जब हम किसी परिणाम को Computer की Screen यानी Output Device पर Display करना चाहते हैं, तब हमें “**stdio.h**” नाम की Header File में Define किए गए **printf()** Function को Use करना होता है।

printf() Function

“सी” भाषा में सभी I/O Functions **stdio.h** नाम की Header File में होते हैं। जब हमे कोई Message या किसी Variable में Stored मान को Screen पर Display करना होता है, तो हम **printf()** Function का प्रयोग करते हैं। इसका Syntax निम्नानुसार है—

```
printf( “ Message CtrlStr1 CtrlStr2 CtrlStrN, Variable1, variable2, variableN);
```

मानलो कि हम एक ऐसा Program बनाना चाहते हैं, जिसे Run करने पर Monitor पर एक String Display हो। चूंकि हम हमारे इस Program में किसी प्रकार का कोई भी Input व Processing नहीं कर रहे हैं, इसलिए इस Program में केवल Output Section ही होगा। यदि हम इस Program का Algorithm बनाना चाहें, तो ये Algorithm निम्नानुसार बनेगा :

Algorithm

```
1  START                               [Algorithm Starts here.]
2  PRINT “Brijvasi”                     [Print the message.]
3  END                                   [Algorithm Ends here.]
```

यदि इस Algorithm के आधार पर हम यदि हम “C” Language में Program बनाना चाहें, तो उस Program का Source Code निम्नानुसार होगा :

```
/* Printing Only One Statement on the screen . */  
  
#include<stdio.h>          /* To Get the Input and Output Services */  
main()                    /* Main Function from where Compiler Executes Program */  
{                          /* Starting of Main Function */  
    printf(" Brijvasi ");  /* Prints the Message */  
}                          /* Ends the Program */
```

इस Program को **Turbo C++** के IDE में एक New File में Type करें और File को **FirstPro.c** नाम से Save करें। इसके बाद File को Compile करके Run करें। File को Compile करने के लिए हम **Ctrl + F9** Key Combination का प्रयोग भी कर सकते हैं। इस Key Combination का प्रयोग करने पर File Compile होकर Run भी हो जाएगी और हमें Output में **Brijvasi** लिखा हुआ Print हो जाएगा।

जैसा कि पहले बताया कि सारे Input/Output Functions “C” की Library की एक Header File **stdio.h** में होते हैं, इसलिए Keyboard से Input लेने या Screen पर Output दर्शाने का काम इसी Header File में Stored Functions के प्रयोग द्वारा सम्पन्न होता है। इसलिए इस Program में “**stdio.h**” नाम की Header File को **#include** किया गया है।

- 1 हर प्रोग्राम में एक **main()** Function होता है। **main()** Function एक Special Function होता है, क्योंकि जब हम “C” Language के किसी Program को Compile करते हैं, तो Compiler सबसे पहले Source Program में **main()** Function को ही खोजता है और Compiler को जहाँ पर **main()** Function मिलता है, Compiler वहीं से Program को Machine Language में Convert करना शुरू करता है।
- 2 {} (**Opening** व **Closing**) Curly Braces के बीच लिखे गए सभी Statements के समूह को **Statement Block** कहा जाता है और इन्हीं Statements का Execution होता है। चूंकि “C” Language में हर Function की शुरुआत एक Opening Curly Brace से व अन्त एक Closing Curly Brace पर होता है, इसलिए किसी भी Program के जितने भी Executable Instructions होते हैं, उन्हें **main()** Function के Statement Block में ही लिखा जाता है।
- 3 “C” Language में हर Statement का अन्त एक Semi Colon द्वारा होता है और “C” में Double Quote के बीच लिखे जाने वाले Statements को **String** कहा जाता है।
- 4 **printf()** Function के “ ” (**Opening** and **Closing**) Double Quotes के बीच लिखा गया Statement Screen पर ज्यों का त्यों Print हो जाता है, क्योंकि ये एक Output Statement है जो किसी Message या मान को Screen पर Display करने का काम करता है।

इस Program को Run करने पर हमें निम्नानुसार Output प्राप्त होता है:

Output

```
Brijvasi
```

इसी Program को यदि चार बार Run किया जाए, तो हमें निम्नानुसार Output प्राप्त होता है :

Output

BrijvasiBrijvasiBrijvasiBrijvasi

ऐसा इसलिए होता है, क्योंकि जब हम दूसरी बार इसी Program को Run करते हैं, तब पिछली बार Run किए गए Program का Output भी हमें फिर से दिखाई देता है। यदि हम चाहें कि हम जितनी बार भी Program को Run करें, हमें पिछली बार का Output Screen पर दिखाई ना दे, तो हमें “conio.h” नाम की Header File में Define किया गया **clrscr()** Function Use करना होता है। जब हम इस Function को Use करते हैं, तो जिस स्थान पर इस Function को Use करते हैं, उस स्थान पर ये Function Screen पर स्थित Message को Clear कर देता है।

Program को Compile व Run करने के लिए हम **Ctrl+F9** Key Combination का प्रयोग करते हैं। लेकिन जब Program को Run किया जाता है, तो Program Result को Monitor पर Display करते ही तुरन्त Terminate हो जाता है और Output को देखने के लिए हमें **Ctrl+F5** Key Combination का प्रयोग करना पड़ता है। यदि हम चाहें कि Program Terminate होने से पहले हमें Program का Output Display करे उसके बाद Terminate हो, तो इस सुविधा को प्राप्त करने के लिए हम **getch()** Function का प्रयोग कर सकते हैं।

getch() Function भी “conio.h” नाम की Header File में ही Define किया गया है। ये Function Keyboard से एक Character को Input के रूप में प्राप्त करने का काम करता है। इसलिए जब हम इस Function को अपने Program में Use करते हैं, तो हमारा Program तब तक रुका रहता है, जब तक कि User Keyboard से कोई Key Press नहीं करता है।

इस स्थिति में यदि हम इस Statement को हमारे Program के अन्तिम Statement के रूप में Use करें, तो हमारा Program तब तक रुक कर Output Display करता रहेगा, जब तक कि User Keyboard से कोई Key Press नहीं कर देता। इन दोनों सुविधाओं को प्राप्त करते हुए यदि हम पिछले Program को Modify करें, तो हम इस Program को निम्नानुसार Modify कर सकते हैं:

```
#include<stdio.h>           //To get Input and Output Services
main()                      //Main Function from where
                            //Compiler Executes Program
{
    clrscr();               //Starting of Main Function
    printf("Gopal & Krishna"); // Clears the Screen
    getch();                // Prints the Name on Screen
                            // To Pause the output screen until we press a key
}
```

Output

Gopal & Krishna

Program Flow

जब इस Program को Run किया जाता है, तब:

- 1^प यदि Program में किसी तरह की कोई Typing Mistake ना हो, तो “C” का Compiler सबसे पहले main() Function को खोजता है।
- 2^प main() Function के मिल जाने के बाद Compiler main() Function के Statement Block में प्रवेश करता है और सबसे पहले **clrscr()** Function को Execute करता है। ये Statement Output Screen को Clear कर देता है।

- 3^ए फिर Program का अगला Statement **printf()** Function Execute होता है, जो Screen पर “Gopal & Krishna” Message को Display करता है।
- 4^ए अन्त में तीसरा Function **getch()** Execute होता है। ये Function User से एक Key Press करने का इन्तजार करता है और जब तक User Key Press नहीं करता है, तब तक वह Output को Screen पर देख सकता है। जैसे ही User Keyboard से किसी Key को Press करता है, Program Terminate हो जाता है।

A Answer the following questions.

- 1 **Data, Processing व Information** को समझाईए तथा इनके बीच के आपसी सम्बंध की व्याख्या कीजिए।
- 2 Computer Program किसे कहते हैं? Program व Software में क्या सम्बंध होता है?
- 3 File किसे कहते हैं? Data File व Program File के बीच क्या अन्तर होता है।
- 4 Application Software व System Software के बीच के अन्तर को स्पष्ट करते हुए दोनों प्रकार के कुछ Software का उदाहरण दीजिए।
- 5 Programming के विभिन्न प्रकारों का वर्णन कीजिए।
- 6 भाषा से आप क्या समझते हैं? Computer किस भाषा को समझता है?
- 7 Computer Languages को कितने भागों में बांटा गया है? वर्णन कीजिए।
- 8 High Level Language व Low Level Languages में अन्तर बताते हुए Assembler, Compiler व Interpreter को समझाईए साथ ही ये भी बताईए कि Assembler, Compiler व Interpreter का मुख्य काम क्या होता है?
- 9 किसी भी Program को Develop करने के विभिन्न Steps को समझाते हुए एक अच्छे Program की विशेषताओं का वर्णन कीजिए।
- 10 Algorithm किसे कहते हैं? दो संख्याओं को गुणा करके तीसरी संख्या का भाग देने का Algorithm बनाईए।
- 11 “C” Language के विकास को बताते हुए “C” Language की विभिन्न Characteristics का वर्णन कीजिए।
- 12 “C” Language के Program का Block Structure बनाकर उसके हर Block को समझाईए।
- 13 “C” Language में Develop किया गया कोई भी Program किस प्रकार से Execute होकर Output प्रदान करता है? Program के Flow को Flow Chart का प्रयोग करते हुए समझाईए।
- 14 Function किसे कहते हैं? Pre-Defined Function व User-Defined Function में क्या अन्तर है?
- 15 Header File से आप क्या समझते हैं? किसी “C” Program में इसकी क्या जरूरत होती है? **stdio.h** Header File को Program में क्यों Include किया जाता है।
- 16 किसी भी Program के मुख्य तीन हिस्से कौन-कौन से होते हैं? यदि किसी Program के तीनों हिस्सों (Input, Process व Output) में से किसी एक हिस्से को क्रम से Program से हटाया जाए, तो हर हिस्से के हटने का Program पर क्या प्रभाव पड़ेगा।
- 17 निम्न Message को Screen पर Display करने का Algorithm बनाईए। इस Algorithm के आधार पर “C” Language में एक Program बनाईए व Program के Flow को समझाईए।

“This is my first program in C Language”

B True/False

- 1 Value or a Set of Values is Data for computer program.

- 2 विभिन्न प्रकार के आंकड़ों (**Data**) का संकलन (**Collection**) करना और फिर उन आंकड़ों को विभिन्न प्रकार से वर्गीकृत (**Classify**) करके उनका विश्लेषण (**Analyze**) करने प्रक्रिया को **Computer** की भाषा में **Data Processing** करना कहा जाता है।
- 3 **Raw Data** व **Processed Data** में कोई अन्तर नहीं होता है।
- 4 जब किसी एक या एक से अधिक समस्याओं को सुलझाने या किसी लक्ष्य को प्राप्त करने के लिए कई स्वतंत्र इकाईयां (**Individual Units**) मिलकर काम कर रहे होते हैं, तो उन इकाईयों के समूह को **System** कहा जाता है।
- 5 **Data File** व **Program File** दोनों में किसी प्रकार का कोई अन्तर नहीं होता है।
- 6 **Instructions** के समूह को **Software** भी कहते हैं।
- 7 **Compiler** व **Antivirus** एक प्रकार के **System Software** के हिस्से होते हैं।
- 8 **Computer Architecture** व **CPU** दोनों को तीन-तीन हिस्सों में विभाजित किया जा सकता है।
- 9 **Programming** तीन तरह की होती है।
- 10 **Compiler** व **Interpreter** दोनों के किसी **Program** को **Machine Language** में **Convert** करने का तरीका अलग-अलग होता है।
- 11 **Computer** एक **Electronic Machine** है, जो केवल **Electrical Signals** की **Binary language** को ही समझता है।
- 12 किसी प्रोग्राम में होने वाली गलतियों को खोजकर उन्हें सही करने की प्रक्रिया को **Bugging** कहते हैं।
- 13 किसी भी समस्या के एक निश्चित समाधान को प्राप्त करने के लिए अनुक्रमिक व चरणबद्ध रूप में अपनाई जाने वाली लिखित प्रक्रिया को हम **एल्गोरिद्म** कहते हैं।
- 14 **High Level Languages** में लिखे जाने वाले **Programs** को एक **Computer Directly Execute** करने में सक्षम होता है।
- 15 “**C**” **Language** एक **Functional Language** है।
- 16 किसी “**C**” **Program** में एक से ज्यादा **main()** **Function** हो सकते हैं।
- 17 **Function** दो तरह के होते हैं, **Pre-Defined** व **User-Defined**
- 18 किसी भी **Computer Program** को **Input**, **Process** व **Output** तीन हिस्सों में बांटा जा सकता है।
- 19 **printf()** **Function** का प्रयोग **Keyboard** से **Data Read** करने के लिए किया जाता है।

BASIC ELEMENTS OF C LANGUAGE

Basic Elements of “C”

“सी” को शुरू करने से पहले इसके कुछ आधारभूत अवयवों को जान लेना बहुत जरूरी है। कुछ खास तरह की **Statements** को लिखने के लिए विभिन्न प्रकार के **Operators** व **Expressions** की जरूरत होती है। हर भाषा में कुछ खास **Statements** व उनको लिखने के कुछ खास तरीके होते हैं। ये ही बात “सी” भाषा पर भी लागू होती है। इस अध्याय में हम “सी” के आधारभूत अवयवों के बारे में जानेंगे।

“C” Characterset

प्रत्येक भाषा में चिन्हों, अंकों, अक्षरों का एक समूह होता है। इन चिन्हों, अंकों व अक्षरों को एक विशेष क्रम में रखने पर एक शब्द बनता है जिसका कि अपना एक खास अर्थ होता है। जैसे र् + अ + म् मिलकर राम शब्द बनाते हैं जिसका अपना एक अर्थ होता है।

इसी तरह “सी” में भी कुछ खास चिन्हों, अंकों व अक्षरों को मान्यता दी गई है, जिनके मिलने से कुछ खास अर्थ निकलते हैं जिन्हे वास्तविक तौर पर सिर्फ कम्प्यूटर ही समझता है। इन चिन्हों, अंकों व अक्षरों के समूह को “सी” भाषा का **“सी” कैरेक्टर सेट** कहा जाता है, जो कि निम्नानुसार होता है:

- 1 Uppercase (A-Z) and Lowercase (a-z) Alphabet
- 2 0 to 9 Digits
- 3 Whitespace Characters (Blank Space, H-Tab, V-Tab, Form Feed, New Line Character, Carriage Return)
- 4 Special Characters

,	Comma	;	Semi Colon
:	Colon	?	Question Mark
.	Dot	'	Single Quote
"	Double Quote		V-Bar
\$	Dollar Sign	#	Pound Sign
&	Ampersand	*	Asterisk
(Left Parentheses)	Right parentheses
[Left Bracket]	Right Bracket
{	Left Curly Brace	}	Right Curly Brace
<	Less Than Sign	>	Greater Than Sign
	Blank	=	Equal to
\	Back Slash	/	Slash
_	Under Score	%	Percent
~	Tilde	^	Upper Carat
+	Plus	-	Minus
!	Exclamation mark		

इस सारणी में हमने जितने भी **Characters** को दर्शाया है, उन सभी **Characters** को हम एक “C” **Program** में समय-समय पर व जरूरत के आधार पर **Use** कर सकते हैं।

“C” Tokens

जिस प्रकार से शब्द, किसी भी पैराग्राफ की वह लघुत्तम इकाई होती है, जिसमें एक विशेष अर्थ विद्यमान रहता है, ठीक इसी तरह इस भाषा में भी ऐसे ही कुछ शब्द, चिन्ह आदि हैं, जो स्वतंत्र रूप

से अपना कुछ अर्थ रखते हैं। "सी" भाषा की वह लघुत्तम इकाई जो स्वतंत्र रूप से अपना कोई अर्थ रखती है, "सी" टोकन् कहलाती है। "सी" भाषा में पांच तरह के "सी" टोकनस् होते हैं, जिन्हे निम्नानुसार समझाया गया है:

Keywords या Reserve Words

"सी" भाषा के कुछ शब्दों को Reserve रखा गया है। इन शब्दों का C Compiler के लिए Special Meaning होता है, इसलिए इन्हें **Keyword** या **Reserve Words** कहते हैं। हर Reserve Word का अपना एक Special Meaning होता है और हर Reserve Word को किसी विशेष परिस्थिति में विशेष काम को पूरा करने के लिए ही Use किया जाता है। हम किसी Reserve Word को किसी सामान्य काम के लिए Use नहीं कर सकते हैं। C भाषा में निम्नानुसार 36 Keywords Define किए गए हैं। कुछ Compilers में इनकी संख्या 32 ही होती है तो कुछ Compilers में इनकी संख्या 36 से ज्यादा भी हो सकती है।

1	auto	2	break	3	case	4	char
5	const	6	continue	7	default	8	do
9	double	10	else	11	enum	12	extern
13	float	14	for	15	goto	16	if
17	int	18	long	19	register	20	return
21	short	22	signed	23	static	24	struct
25	switch	26	typedef	27	union	28	unsigned
29	void	30	while	31	asm	32	fortran
33	pascal	34	huge	35	far	36	near

Identifiers – Constant and Variable Name

जब हम Program Develop करते हैं, तब हमें विभिन्न प्रकार के Data को Computer की Memory में Input करके उस पर विभिन्न प्रकार की Processing करनी होती है। Computer में Data के साथ हम चाहे किसी भी प्रकार की प्रक्रिया करना चाहें, हमें हर Data को सबसे पहले Computer की Memory में Store करना जरूरी होता है। Computer की Memory में किसी Data को Store किए बिना हम उस Data के साथ किसी प्रकार की कोई प्रक्रिया नहीं कर सकते हैं।

Computer में Memory के हर Location का एक **Unique Address** होता है। जब हम Computer में किसी Data को Process करने के लिए Input करते हैं, तब वह Data Memory के किसी ना किसी Location पर जाकर Store हो जाता है।

लेकिन हमें कभी भी सामान्य तरीके से ये पता नहीं चल सकता है कि हमारे द्वारा Input किया गया Data Computer की किस Memory Location पर Store हुआ है और ना ही हम स्वयं कभी ये तय कर सकते हैं कि हमारा Data किस Memory Location पर Store होगा, क्योंकि Data को Memory Allocate करने का काम अपनी सुविधानुसार हमारा Operating System स्वयं करता है।

जिस समय हमारे Data को Store करने के लिए Compiler Memory Reserve करता है, उसी समय हम उस Reserve होने वाली Memory Location का एक नाम Assign कर देते हैं। इस नाम के द्वारा ही हम हमारे Data को Computer की Memory में Identify कर सकते हैं। हमारे द्वारा किसी Data की Memory Location को दिए जाने वाले इस नाम को ही **Identifier** कहते हैं।

हम किसी Memory Location का जो नाम Assign करते हैं, उन नामों को कुछ नियमों को ध्यान में रख कर परिभाषित करना होता है, क्योंकि "सी" कम्पाइलर उन विशेष प्रकार के नियमों के आधार पर

परिभाषित किये गए नामों के साथ ही विभिन्न प्रकार की प्रक्रियाएं करता है। किसी Identifier को नाम देने के लिए हमें निम्न नियमों को Follow करना होता है, जिन्हें **Identifier Naming Convention** कहा जाता है:

- किसी भी Identifier के नाम में किसी भी **Upper Case** व **Lower Case Character** का प्रयोग किया जा सकता है।
- किसी भी Identifier के नाम में **Underscore** का भी प्रयोग किया जा सकता है।
- किसी भी Identifier के नाम में यदि हम अंकों का प्रयोग करना चाहें, तो अंकों का प्रयोग करने से पहले कम से कम एक **Character** या **Underscore** का होना जरूरी होता है।
- इसके अलावा Identifier के नाम में किसी भी प्रकार के **Special Symbol** जैसे कि **Period, Comma, Blank Space** आदि का प्रयोग नहीं किया जा सकता है। साथ ही हम Identifier के नाम में किसी **Reserve Word** या किसी **Built-In Function** के नाम का प्रयोग भी नहीं कर सकते हैं।
- किसी भी नाम की शुरुआत किसी अंक से नहीं हो सकती है।
- "सी" एक **Case Sensitive Language** है, इसलिए इस भाषा में **Capital Letters** व **Small Letters** के नाम अलग-अलग माने जाते हैं। जैसे **int Sum** व **int sum** दो अलग-अलग **Variable Name** या **Identifiers** होंगे ना कि समान।

किसी **Variable Identifier** या **Constant Identifier** का हम निम्न तरीके का कोई भी नाम रख सकते हैं, जो कि "C" के **Naming Rules** का पूरी तरह से पालन करते हैं:

```
number
number2
amount_of_sale
_amount
salary
daysOfWeek
monthsOfYear
```

लेकिन आगे दिए जा रहे नाम गलत हैं और हम इन तरीकों के नाम किसी **Variable** या **Constant Identifier** को **Assign** नहीं कर सकते हैं, क्योंकि ये नाम "C" Language के **Naming Rules** का पूरी तरह से पालन नहीं करते हैं:

```
number#           /* illegal use of Special Symbol # */
number2*          /* illegal use of Special Symbol * */
1amount_of_sale   /* Name could not start with a Digit */
&$amount          /* illegal use of Special Symbol & and $ */
days Of Week     /* illegal use of Special Symbol Blank Space between name */
months OfYear10  /* illegal use of Special Symbol Blank Space between name */
```

Exercise

1 Specify invalid variable names and give proper reason why they are invalid?

TOTALPERCENT	_BASIC	basic-salary	1 st value
\$per#	daysIn1Year	LeAPyEAR	432
float	Integer	longInteger	hours.
daysInWeek	Book Name	population	day of week
minute.	father's Name	2910_India	_total_days_

- 2 Keyword किसे कहते हैं ?
- 3 Identifiers से आप क्या समझते हैं ? **Keywords** व **Identifiers** में क्या अन्तर है ?
- 4 *Identifier Naming Convention* से आप क्या समझते हैं ?
- 5 Identifiers Create करते समय हमें किन नियमों को ध्यान में रखना जरूरी होता है ?

Constants and Variables

सभी Programming Languages में यदि कोई चीज Common होती है, तो वह यही है कि सभी Programming Languages में Develop किए जाने वाले Programs में Data को Input किया जाता है और उन पर Required Processing Perform करके Output Generate किया जाता है।

चूंकि किसी भी Computer Program में सबसे Important चीज Data ही होती है, इसलिए हर Computer Program में इसी बात का ध्यान रखा जाता है कि Data को विभिन्न तरीकों से Store किया जाए, ताकि उन पर विभिन्न प्रकार की Processing को Apply करके विभिन्न प्रकार के Results Generate किए जा सकें। Data Memory में किस प्रकार से Store होंगे और किस प्रकार से उन पर Processing को Apply किया जाएगा, इस बात का Track रखने के लिए Programs में **Constants** व **Variables** का प्रयोग किया जाता है।

Constants

किसी भी Computer Program में हम विभिन्न प्रकार के मानों को Computer में Store करते हैं, उन्हें Manage करते हैं, उन पर Required Processing Apply करते हैं और उनके परिणाम को Output में प्राप्त करते हैं। यदि हम Real World में देखें तो दो तरह के मान होते हैं। एक मान वे होते हैं जिन्हें कभी Change नहीं किया जाता है।

जैसे कि साल में कुल 12 महीने होते हैं। इन महीनों की संख्या हमेशा निश्चित होती है। कभी भी किसी भी साल में 11 या 13 महीने नहीं हो सकते। इसी तरह से हर महीने का एक निश्चित नाम होता है। हर Week में सात दिन होते हैं। हर दिन का एक निश्चित नाम होता है। इसी तरह से PI का मान 22/7 होता है।

हम समझ सकते हैं कि ऐसी ही हजारों चीजें हैं, जिनके मान हमेशा निश्चित होते हैं। जो मान हमेशा निश्चित होते हैं, उन मानों को Hold करने वाले Identifiers को **Constants** कहा जाता है। इसी तरह से किसी Computer Program में Declare किया गया वह Identifier जो ऐसे ही किसी Constant मान को Hold करता है और पूरे Program में अपने Data को Change नहीं करने देता है, Constant कहलाता है।

हम किसी भी Data को मान या मानों के एक समूह के रूप में मान सकते हैं। यानी किसी भी तथ्य को Computer Program में Represent करने के लिए हमें उस तथ्य को किसी ना किसी मान के रूप में परिभाषित करना होता है। Computer में मानों को या तो Texts के रूप में Represent किया जाता है या फिर किसी अंक के रूप में।

उदाहरण के लिए यदि हमें साल के कुल महीनों को Computer में Store करना हो तो हम अंक 12 को उपयोग में लेते हैं, जो कि एक संख्या है। जबकि यदि हमें किसी महीने के नाम माना "January" को Computer में Store करना हो तो हम Characters के समूह का प्रयोग करते हैं।

इस उदाहरण के आधार पर हम कह सकते हैं कि किसी भी Real World मान को Computer में या तो किसी अंक या अंकों के समूह के रूप में Define किया जाता है या किसी Character या Characters के समूह के रूप में।

विभिन्न अंक या अंकों के समूह को हम **Numeral Constants** कह सकते हैं और विभिन्न Characters व Characters के समूह को **Character** या **String Constants** कह सकते हैं। उदाहरण के लिए मान लो कि हमें 100 रुपये का 6.0 प्रतिशत की दर से ब्याज ज्ञात करना है। ये Calculation हम निम्नानुसार Perform कर सकते हैं:

```
Interest = 100 * 6.0 / 100
```

इस Statement में Numerical मान 100 व 6.0 स्थिर मान हैं, इसलिए इन्हें **Constant** कहा जाता है। मानलो कि हमें किसी Program में इस Calculation को कई बार Perform करना पड़ता है। इस स्थिति में हम इस Statement को पूरे Program में कई बार लिख सकते हैं।

लेकिन थोड़े समय बाद यदि हमें 6.0 के बजाय 7.0 प्रतिशत की दर से ब्याज Calculate करने के लिए इसी Program को Modify करना पड़े, तो हमने Program में जितनी बार इस Calculation को Perform किया है, उतनी ही बार अंक 6.0 के स्थान पर 7.0 को Replace करना पड़ेगा।

यदि हमने हमारे Program में 200 बार इस Statement को Use किया गया हो तो हमें हमारे Program में 200 स्थानों पर 6.0 के स्थान पर 7.0 करना पड़ेगा जो कि काफी असुविधाजनक काम होगा। क्योंकि Program को Modify करने में भी काफी समय लगेगा और गलतियां होने की भी काफी सम्भावना रहेगी, क्योंकि पूरे Program में किसी एक भी स्थान पर यदि हमने 6.0 को 7.0 से Replace नहीं किया, तो Program का Output गलत ही आएगा।

इस प्रकार की स्थितियों को Avoid करने के लिए Programmers हमेशा कुछ Symbolic Constants का प्रयोग करते हैं, जो सामान्यतया वे शब्द होते हैं, जो Program में किसी मान को Represent करते हैं।

यदि हम हमारे इस पिछले Expression की ही बात करें, तो 6.0 को Represent करने के लिए हम **PERCENT** नाम का एक Symbolic Content Use कर सकते हैं, जो Current Percent को Represent करता है और Program की शुरुआत में इस Percent को वह दर प्रदान कर सकते हैं, जिसे पूरे Program में Calculate करना है।

"C" Language में किसी Constant को Represent करने के लिए जो Statement लिखा जाता है, उसे Constant Declare करना कहते हैं और इसे निम्नानुसार Declare करते हैं:

```
const float PERCENT = 6.0;
```

“C” में **const** Keyword का प्रयोग तब किया जाता है, जब हमें “C” Compiler को ये बताना होता है, कि हम जिस Identifier द्वारा किसी मान को Program में Represent कर रहे हैं, उस Identifier के मान में पूरे Program के दौरान किसी प्रकार का Change नहीं किया जा सकता है।

इसी तरह से **float** Keyword “C” Compiler को ये बताता है कि हम जिस Constant मान को Store करना चाहते हैं, वह मान एक Floating Point मान या दसमलव वाला मान है। PERCENT शब्द एक Symbolic Content है और इस Expression में = (**Equal To**) का चिन्ह बताता है कि = के Left Side में जो Word है वह Word उस मान के बराबर है जो = चिन्ह के Right Side में है जो कि हमारे इस Statement में 6.0 है।

यानी हम इस Calculation में 6.0 लिखें या PERCENT लिखें, दोनों से निकलने वाला परिणाम समान ही प्राप्त होगा, क्योंकि दोनों ही समान मान को Represent कर रहे हैं।

```
Interest = 100 * PERCENT / 100;
```

सामान्यतया Symbolic Constants को Program के अन्य Codes से अलग दिखाने के लिए UPPERCASE Letters में लिखा जाता है।

Variables

Program के वे मान जो पूरे Program में समय-समय पर आवश्यकतानुसार बदलते रहते हैं, Variables कहलाते हैं। Variables कभी भी किसी स्थिर मान को Represent करने के लिए Use नहीं किए जाते हैं। जब भी हमें किसी Constant को Program में Use करना होता है, तो उस Constant को Represent करने के लिए हमें Symbolic Constants की जरूरत होती है। इन Symbolic Constants को ही Literal भी कहा जाता है।

सवाल ये पैदा होता है कि Program में Variables की क्या जरूरत है ? इसे समझने के लिए पिछले Statement को ही लेते हैं, जो कि निम्नानुसार है:

```
Interest = 100 * PERCENT / 100;
```

इस Statement में Interest एक **Variable** है। यानी किसी Calculation के Result को Store करने के लिए हमें हमारे Program में हमेशा एक ऐसी Memory की जरूरत होती है, जिसमें विभिन्न प्रकार के बदलते हुए मान Store हो सकें। इस Statement द्वारा हम केवल 100 का ही PERCENT ज्ञात कर सकते हैं।

लेकिन सामान्यतया हमें अलग-अलग स्थानों पर अलग-अलग प्रकार के मानों का Percent ज्ञात करना होता है। ऐसे में हर संख्या का Percent ज्ञात करने के लिए यदि हमें अलग से Program बनाना पड़े तो ये एक बहुत ही असुविधाजनक बात होगी।

Program ऐसा होना चाहिए कि किसी एक ही Program से एक प्रकार से Perform होने वाली विभिन्न प्रकार की Calculations को Perform किया जा सके। यानी हम यदि 100 की जगह 200 कर दें, तो हमें 200 का Interest प्राप्त हो जाए। यदि हम Program को Multipurpose बनाना चाहते हैं, तो हमें 100 को भी किसी Symbolic तरीके से Represent करना होगा। ये काम हम निम्नानुसार Statement द्वारा कर सकते हैं:

```
Principal = 100;
Interest = Principal * PERCENT / 100;
```

हम देख सकते हैं कि यदि Principal का मान 100 से 200 कर दिया जाए तो Interest नाम के Variable में हमें Principal 200 का Interest प्राप्त होगा। चूंकि मूलधन 100 के Symbolic Representative Principal का मान बदल कर 200, 300, 400 आदि किया जा सकता है, इसलिए **Principal** भी एक Variable है और Principal के Change होने से Calculate होने वाले Interest में भी परिवर्तन होता है, इसलिए **Interest** भी एक Variable है।

वास्तव में Constant Identifier व Variable Identifier के नाम में किसी प्रकार का कोई अन्तर नहीं होता है। अन्तर केवल इनके Declaration के तरीके में होता है। हम Variable Identifier को Declare करें या Constant Identifier को, दोनों ही स्थितियों में हमें Identifier Naming Convention के उपरोक्त सभी नियमों का पालन करना होता है।

Identifier Declaration

किसी भी प्रकार के Data को Process करने के लिए हमें सबसे पहले ये तय करना होता है, कि हम किस प्रकार के Data को Computer की Memory में Store करना चाहते हैं, क्योंकि जब तक हम Process किए जाने वाले Data को Computer की Memory में Store नहीं कर देते हैं, तब तक हम उस Data को Process नहीं कर सकते हैं।

चूंकि अलग-अलग प्रकार के Data Memory में अलग-अलग Size की Space Reserve करते हैं, इसलिए जब हमें Process किए जाने वाले Data के Type का पता चल जाता है, तब हम उस Data Type को Represent करने वाले Keyword के आधार पर Memory में कुछ Space Reserve करते हैं और उस Space का कोई नाम Assign करते हैं। ये नाम उस Reserved Memory Location का एक Symbolic Identifier होता है, जो कि *Identifier Naming Convention* के नियमों के आधार पर तय किया जाता है।

Program की जरूरत के आधार पर किसी Data को Store करने के लिए Computer की Memory में Space Reserve करने व उस Space का कोई Symbolic नाम देने की प्रक्रिया को **Identifier Declaration** कहते हैं।

यदि Define किए जाने वाले Identifier का मान पूरे Program में स्थिर रहे, तो इस प्रक्रिया को **Constant Declaration** कहते हैं, जबकि यदि Define किए जाने वाले Identifier का मान पूरे Program में समय-समय पर Program की जरूरत के आधार पर बदलता रहे, तो इसे **Variable Declaration** कहते हैं।

Identifier Declaration के समय हमें हमेशा दो बातें तय करनी होती हैं। पहली ये कि हमें किस प्रकार (**Data Type**)का Data Computer की Memory में Store करना है और दूसरी ये कि Reserve होने वाली Memory Location को क्या नाम (**Identifier Name**) देना है। Identifier Declare करने का General Syntax निम्नानुसार होता है:

Syntax:

```
DataTypeModifier DataType IdentifierName;
```

Data Type Modifier

Syntax के इस शब्द के स्थान कुछ ऐसे Keywords का प्रयोग किया जाता है, जिनका प्रयोग करके **Data Type** की किसी मान को Store करने की क्षमता को बढ़ाया या घटाया जा सकता है। इस शब्द के स्थान पर जरूरत के आधार पर **short**, **long**, **signed** या **unsigned** Keywords का प्रयोग

किया जा सकता है। यहां Use किया जाने वाला Keyword Optional होता है। यदि हमें जरूरत ना हो तो हम इसका प्रयोग किए बिना भी Memory Create कर सकते हैं।

Data Type

Syntax के इस शब्द के स्थान पर कुछ ऐसे Keywords का प्रयोग किया जाता है, जो ये तय करते हैं कि हम Computer की Memory में किस प्रकार के Data को Store करना चाहते हैं। उदाहरण के लिए यदि हमें केवल पूर्णांक संख्याओं को Store करने के लिए Memory Reserve करना हो, तो हम इस शब्द के स्थान पर **int** Keyword का प्रयोग करते हैं, जबकि यदि हमें किसी दसमलव वाली संख्या के लिए Memory Reserve करना हो, तो हमें इस शब्द के स्थान पर **float** Keyword को Use करना होता है।

Identifier Name

Syntax के इस शब्द के स्थान पर हम "C" के Identifier Naming Convention के नियमों के आधार पर Reserve होने वाली Memory Location का एक Symbolic नाम Specify किया जाता है। हम Reserve किए गए Memory Location पर स्थित जिस मान को पूरे Program में Access करना चाहते हैं, उस मान को हम इसी Symbolic नाम से Access करते हैं।

मानलो कि हम किसी Student की **Age** को Computer में Store करना चाहते हैं। चूंकि Age एक प्रकार का Numerical पूर्णांक मान होता है, इसलिए हमें इस Integer Data Type के मान को Store करने के लिए "C" Language के **int** Keyword का प्रयोग करना होता है।

"C" Language में सभी प्रकार के Numerical मान Positive Signed मान होते हैं, जिसमें Minus की संख्या को भी Store किया जा सकता है। लेकिन चूंकि Age कभी भी Minus में नहीं हो सकती है, इसलिए हमें **int** Data Type से पहले हमें **unsigned** Modifier का प्रयोग करना होगा।

चूंकि हम Student की Age को Computer में Store करने के लिए Unsigned Integer प्रकार की Memory Location को Reserve कर रहे हैं, इसलिए इस Memory Location को Identify करने के लिए हम Symbolic नाम के रूप में **studentAge** शब्द का प्रयोग कर सकते हैं, जो कि **Identifier Naming Convention** के नियमों के आधार पर पूरी तरह से सही है।

इस Discussion के आधार पर यदि हम Student की **Age** को Store करने के लिए एक Identifier Create करें, तो हमें "C" Language में निम्नानुसार Statement लिखना होगा:

Variable Declaration

```
unsigned int studentAge;
```

जहां **unsigned** Keyword Modifier है। **int** Data Type है और **studentAge** Reserve होने वाली Memory Location का Symbolic नाम है। अब यदि हम Student की Age को Reserve होने वाली Memory Location पर Store करना चाहें, तो हमें निम्नानुसार Statement लिखना होता है:

```
studentAge = 21;
```

इस Statement द्वारा उस Memory Location पर Integer मान 21 Store हो जाता है, जिसका नाम **studentAge** है। यदि हम चाहें तो इस Memory Location पर 21 के स्थान पर अगले Statement में निम्नानुसार 23 भी कर सकते हैं:

```
studentAge = 32;
```

यानी हम जितनी बार चाहें, उतनी बार अपनी जरूरत के आधार पर *studentAge* Symbolic Name वाली Memory Location का मान Change कर सकते हैं। इसलिए इस Symbolic Identifier को हम **Variable** भी कह सकते हैं। लेकिन यदि हम चाहते हैं, कि *studentAge* में केवल एक ही बार किसी Integer मान को Store किया जाए और किसी भी स्थिति में पूरे Program में *studentAge* के मान को Change ना किया जा सके, तो हमें इसी Declaration के समय निम्नानुसार **const** Keyword का प्रयोग करना होता है:

Constant Declaration

```
const unsigned int studentAge = 21;
```

जब हम इस प्रकार से किसी Identifier को Declare करते हैं, तब इस प्रकार के Declaration को Constant Identifier Declaration कहते हैं। हम जब भी कभी किसी Identifier को Constant Declare करते हैं, तो उस Identifier को Declare करते समय ही हमें Symbolic Name की Memory Location पर Store होने वाले मान को भी Specify करना जरूरी होता है। क्योंकि Constant Identifier हम उसी स्थिति में Declare करते हैं, जब हमें पता होता है कि किसी Constant Identifier का मान पूरे Program में क्या होना चाहिए।

जब हम किसी Identifier को Constant Declare करते हैं, तब यदि हम Program में किसी Statement द्वारा उस Constant के मान को Change करने की कोशिश करते हैं, तो Compiler हमें ऐसा नहीं करने देता है। यानी यदि हम उपरोक्त Statement लिखने के बाद निम्नानुसार दूसरा Statement लिखें और Program को Compile करें, तो Compiler हमें निम्नानुसार Error Message प्रदान करता है:

```
studentAge = 23; // Error: Cannot modify a const object.
```

इसी तरह से यदि हम किसी const Identifier को Declare करते समय उसे कोई मान प्रदान ना करें, यानी Constant को निम्नानुसार Declare करें:

Constant Declaration

```
const unsigned int studentAge;
```

तो इस स्थिति में हमें निम्नानुसार Error Message प्राप्त होता है:

```
Error: Constant variable 'studentAge' must be initialized
```

Initialization

हम किसी भी Identifier को उसके Declaration के समय ही किसी ना किसी प्रकार का मान भी प्रदान कर सकते हैं। Identifier को उसके Declaration के समय ही कोई मान प्रदान करने की प्रक्रिया को **Value Initialization** करना कहते हैं। जैसे :-

```
int digit1 = 12;  
int digit2 = 33;
```

हम एक ही समय में एक से अधिक Identifiers को जो कि समान प्रकार के Data Type के हों, Declare कर सकते हैं व किसी ना किसी मान से Initialize भी कर सकते हैं। जैसे:

```
int digit1, digit2 ; OR
```

```
int digit1 = 12, digit2 = 33 ;
```

Expressions

जब दो या दो से अधिक **Operands** पर **Operators** की सहायता से कोई प्रक्रिया करके कोई परिणाम प्राप्त करना होता है, तो उस स्थिति में हम जो **Statement** लिखते हैं, उसे **Expression** कहते हैं। उदाहरण के लिए दो संख्याओं को जोड़ कर प्राप्त मान को किसी तीसरे Identifier में Store करने के लिए हम निम्न Statement लिखते हैं:

```
sum = digit1 + digit2
```

इस Statement को Expression कहा जाता है। उपरोक्त Statement एक Arithmetical Expression का उदाहरण है। इसी तरह विभिन्न प्रकार के Logical, Relational आदि Operators को Use करके विभिन्न प्रकार के Expressions बनाए जा सकते हैं।

Exercise:

- 1 Variable व Constant के बीच के अन्तर को Explain कीजिए।
- 2 Identifier Declaration क्यों किया जाता है? Identifier Declare करने के लिए हमें किस Syntax का प्रयोग करना पड़ता है? इस Syntax को समझाईए।
- 3 Variable Identifier व Constant Identifier में अन्तर स्पष्ट करते हुए Integer प्रकार के दो Variable व Constant Declare कीजिए।
- 4 एक उदाहरण देते हुए Initialization व Expression के अन्तर को स्पष्ट कीजिए।

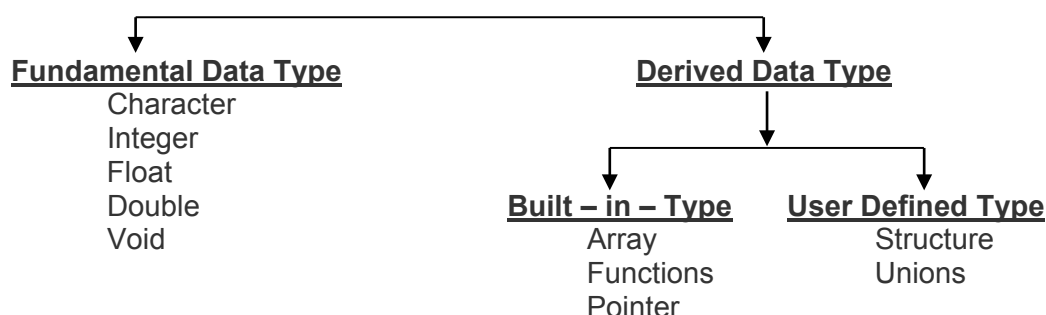
Data and Data Types

मान या मानों के समूह Computer के लिए Data होता है। Real World में भी Data (मान या मानों का समूह [Value or a Set of Values]) कई प्रकार के होते हैं। जैसे किसी व्यक्ति की उम्र को हम संख्या के रूप में दिखाते हैं, जबकि उस व्यक्ति के नाम को Characters के समूह के रूप में परिभाषित करते हैं। इसी Concept के आधार पर “C” Language में भी विभिन्न प्रकार के Data को Store करने के लिए विभिन्न प्रकार के Data Types के Keywords को Develop किया गया है।

यदि गौर किया जाए तो वास्तव में Data केवल दो तरह के ही होते हैं। या तो Data Numerical होता है, जिसमें केवल आंकिक मान होते हैं और इनके साथ किसी ना किसी प्रकार की Calculation को Perform किया जा सकता है या फिर Alphanumerical जो कि Characters का समूह होते हैं, जिनके साथ किसी प्रकार की किसी Calculation को Perform नहीं किया जा सकता।

“C” Language में भी Data को Store करने के लिए दो अलग तरह के Data Types में विभाजित किया गया है, जिन्हें क्रमशः Primary (Standard) Data Type व Secondary (Abstract or Derived) Data Type कहा जाता है। Primitive Data Type **Standard Data Type** होते हैं, जबकि Derived या Abstract Data Type Primitive Data Type पर आधारित होते हैं। फिर जरूरत के अनुसार इन दोनों Data Types को भी कई और भागों में बांटा गया है, जिन्हें हम निम्न चित्र द्वारा समझ सकते हैं:

“सी” Data Type



किसी भी Identifier, या Constant को Define करने से पहले यह निश्चित करना जरूरी होता है कि वह Identifier (Variable, Array, Constant आदि) किस तरह का मान Store करेगा।

यानी यदि हमें छोटी संख्या को Computer में Store करना हो तो हमें **int** प्रकार के Data Type के Identifier को Use करना होता है, जबकि यदि हमें बड़ी संख्या को Computer में Store करना हो तो हमें **long** प्रकार के Data Type का Identifier Use करना होता है।

इसी तरह से यदि हमें Computer में केवल एक Character को Store करना हो, तो हम **char** प्रकार के Data Type को Use करते हैं, जबकि यदि हमें पता ही ना हो कि हम किस प्रकार के Data को Computer में Store करेंगे, तो हम **void** प्रकार का Identifier Declare करते हैं। यानी हमें किस प्रकार के मान को किस प्रकार के Data Type के Identifier में Store करना है, यह बात Program की Requirement पर निर्भर करता है।

“C” Language हमें विभिन्न प्रकार के Data Types को अलग-अलग Store व Manage करने की सुविधा इसलिए Provide करता है, ताकि सारा काम Systematically हो सके और प्रोग्राम को विश्वसनीय व तेजी से Run होने वाला बनाया जा सके। “सी” भाषा में कुल चार तरह के Fundamental Data Types हैं, जिन्हे निम्नानुसार समझाया गया है:

Integer

जब प्रोग्राम में सिर्फ पूर्णांक संख्याओं को Store करने के लिए ही Memory Reserve करनी होती है, तब Identifier को Integer प्रकार का Declare किया जाता है। इसमें भिन्नांक संख्याएं नहीं हो सकती है। किसी Variable को Integer प्रकार का Declare करने के लिए Identifier के नाम के साथ **int** Keyword का प्रयोग करके “C” Compiler को बताया जाता है कि वह Identifier केवल पूर्णांक संख्याओं को ही Memory में Store कर सकेगा। int प्रकार के Identifier में हम + व - दोनों तरह के मान रख सकते हैं।

जब हम किसी Identifier को Declare करते समय किसी भी Modifier का प्रयोग नहीं करते हैं, तब Create होने वाला Identifier By Default **signed** होता है। इसलिए यदि हमें ऐसे मान को Store करने के लिए Identifier Create करना हो, जो Positive या Negative किसी भी प्रकार के मान को Store कर सकता है, तो हमें **signed** Keyword का प्रयोग करने की जरूरत नहीं होती है, लेकिन यदि हमें केवल Positive मान को Store करने के लिए ही Identifier Create करना हो, तो उस स्थिति में हमें Data Type के साथ **unsigned** Modifier का प्रयोग करना जरूरी होता है। “सी” में Integer प्रकार के Identifier को निम्न भागों में बांटा गया है:

int OR signed int

यदि हम 16 – Bit Compiler का प्रयोग करते हैं, तो इस प्रकार का Identifier Memory में दो Byte का Storage Space Reserve करता है जबकि यदि हम 32 – Bit का Compiler Use करते हैं तो इस प्रकार का Identifier Memory में 4 Bytes का Storage Space Reserve करता है। int प्रकार का Identifier 16 – Bit Compiler Use करने पर –32768 से 32767 मान तक की संख्या को स्टोर कर सकता है।

जब Identifier के साथ – चिन्ह होता है या संख्या का मान ऋणात्मक हो सकता है, तो Identifier के साथ **signed int** लिखते हैं। जैसे **signed int total** लेकिन यदि हम **signed** Key word का प्रयोग नहीं करते हैं तो भी int प्रकार का Identifier Negative Values को Hold कर सकता है। इस प्रकार के Data को Memory Space Allocate करने के लिए हमें निम्नानुसार Declaration करना होता है:

```
signed int bankDues; or int bankDues;
```

unsigned int

यदि हम 16 – Bit Compiler का प्रयोग करते हैं तो इस प्रकार का Identifier Memory में दो Byte का Storage Space Reserve करता है जबकि यदि हम 32 – Bit का Compiler Use करते हैं तो इस प्रकार का Identifier Memory में 4 Bytes का Storage Space Reserve करता है। इसमें int Keyword के पहले **unsigned** Keyword Use किया जाता है।

unsigned int प्रकार का Identifier 0 से 65535 तक के मान की संख्या को स्टोर कर सकता है, क्योंकि इसमें हमेशा एक Positive पूर्णांक मान ही Store हो सकता है। इसलिए इसकी कुल क्षमता $32768 + 32767 = 0$ से 65535 तक की संख्या Store हो जाती है। इस प्रकार के Data को Store करने के लिए हमें निम्नानुसार Declaration करना होता है:

```
unsigned int villagePopulation;
```

short OR signed short int

जब हमें काफी छोटी संख्या को Store करने के लिए Memory Reserve करनी होती है और Store की जाने वाली संख्या पूर्णांक ही होती है, तब Identifier के नाम के साथ **short** Modifier का प्रयोग करते हैं। यदि हम **signed** Modifier का प्रयोग ना भी करें, तो भी हम इस प्रकार के Identifier में Minus Sign के मान को Store कर सकते हैं। लेकिन यदि हम चाहें तो **short** Keyword के साथ **signed** Modifier का प्रयोग भी कर सकते हैं। इस प्रकार का Identifier मेमोरी में हमेशा 2 Byte की Storage Space Reserve करता है, फिर चाहे हम 16 – Bit Compiler Use कर रहे हों या 32 – Bit Compiler Use कर रहे हों।

सामान्यतया **short** प्रकार के Identifier का प्रयोग तब किया जाता है, जब हम 32 – Bit Compiler को Use कर रहे होते हैं और हमें छोटे मान को Store करना होता है। चूंकि 32 – Bit Compiler में Integer 4 Byte का होता है, इसलिए छोटी Integer संख्याओं को Store करने के लिए हम **short** प्रकार के Identifier को Use करके 2 Bytes की बचत कर सकते हैं। 16 – Bit Compiler में हम चाहे **int** Type का Identifier Declare करें या **short** Type का, दोनों ही Memory में 2 Byte का ही Storage Space Reserve करते हैं। इस प्रकार के Identifier को हम निम्नानुसार Declare कर सकते हैं:

signed short int *normalTemperature*; or
short int *normalTemperature*;

short Modifier का प्रयोग हमेशा Integer प्रकार के Identifier के साथ ही किया जाता है, इसलिए यदि हम चाहें तो उपरोक्त दोनों Declaration निम्नानुसार बिना **int** Keyword का प्रयोग किए हुए भी कर सकते हैं:

signed short *normalTemperature*; or
short *normalTemperature*;

unsigned short int

जब हम 16 – Bit Compiler को Use करते हैं और हमें छोटी लेकिन केवल Positive संख्या को ही Computer में Store करने के लिए Storage Space Reserve करना होता है, तब हम **unsigned short int** प्रकार का Identifier Declare करते हैं। ये Identifier भी Memory में 2 Bytes की Storage Space ही Reserve करता है और इसे Declare करने का तरीका **signed short** Type के Identifier को Declare करने के समान ही होता है। यानी:

unsigned short int *salary*; or
unsigned short *salary*;

long OR signed long int

जब हमें काफी बड़ी संख्या का प्रयोग करना होता है तब हम इस Data Type का चयन करते हैं। यह मेमोरी में 4 Byte की Storage Space Reserve करता है और -2,147,483,648 से 2,147,483,647 मान तक की संख्या को Store कर सकता है। इस Data Type का प्रयोग अक्सर वैज्ञानिक गणनाओं में किया जाता है जहां काफी बड़ी संख्याओं की गणना करनी होती है। इस प्रकार का Identifier Declare करने के लिए हम निम्न में से किसी भी तरीके को Use कर सकते हैं:

long *velocity*;
long int *velocity*;
signed long *velocity*;
signed long int *velocity*;

long प्रकार का Identifier हमेशा By Default Integer प्रकार का ही होता है, इसलिए यदि हम **long** के साथ **int** Keyword प्रयोग नहीं भी करते हैं, तब भी Create होने वाला Identifier **long int** प्रकार का ही होता है।

unsigned long int

जब हमें काफी बड़ी लेकिन केवल धनात्मक संख्या को Store करने के लिए ही Memory में Storage Space Create करना होता है, तब हम इस Data Type को Use करते हैं। यह भी मेमोरी में 4 Byte की Storage Space Reserve करता है, लेकिन इस प्रकार के Identifier में Minus की संख्या को Store नहीं किया जा सकता है। इस प्रकार के Data Type के Identifier की संख्या Store करने की Limit 0 से 4,294,967,295 है। इस प्रकार का Identifier Define करने के लिए हम निम्न तरीकों से Statements को Use कर सकते हैं:

unsigned long *population*;

```
unsigned long int population;
```

Float

जब हमें प्रोग्राम में भिन्नात्मक व दशमलव वाली संख्याओं को Store करने के लिए Memory की जरूरत होती है, तब हम float प्रकार का Identifier Declare करते हैं। ये Identifier मेमोरी में 4 Bytes की Storage Space Reserve करता है और भिन्न या घातांक रूपों में $3.4E-38$ से $3.4E+38$ मान तक की संख्या को Store कर सकता है। इस प्रकार के Identifier के साथ **unsigned, signed, short** या **long** किसी भी Modifier का प्रयोग नहीं किया जा सकता है। इस तरह का Identifier निम्नानुसार तरीके से Declare कर सकते हैं:

```
float lightSpeed;
```

जब हम किसी Float प्रकार के Variable में कोई मान Store करना चाहते हैं, तब मान के साथ हमें **f** या **F** Character को Post-Fix के रूप में Use करना जरूरी होता है। यदि हम ऐसा नहीं करते हैं, तो Float प्रकार के Identifier में Store होने वाला मान Float प्रकार का नहीं बल्कि Double प्रकार का होता है। यानी "C" Language में हर Real Number *By Default* Double प्रकार का होता है। इसलिए यदि हम lightSpeed Variable में कोई मान Store करना चाहें, तो हमें ये मान निम्नानुसार Statement द्वारा Store करना होगा:

```
lightSpeed = 38000000000f;      OR
lightSpeed = 38000000000F;
```

Double

जब हमें प्रोग्राम में इतनी बड़ी भिन्नात्मक या घातांक संख्या के साथ प्रक्रिया करनी होती है, जो की **float** की Range से भी ज्यादा हो, तब हम इस Data Type का प्रयोग करके Identifier Declare करते हैं। इन्हें भी हम दो भागों में बांट सकते हैं:

Double

ये मेमोरी में 8 Byte की Storage Space Reserve करता है और $1.7E-308$ से $1.7E+308$ मान तक की संख्या को Store कर सकता है। इस प्रकार के Identifier को हम निम्नानुसार Declare कर सकते हैं:

```
double lightMovementIn1Year;
```

long double

जब Double के साथ **long** Key word लगा दिया जाता है यानी जब **long double** प्रकार का Data Type Use करते हैं तब वह Identifier बड़ी से बड़ी संख्या को Store कर सकता है। यह मेमोरी में 10 Byte की Storage Space Reserve करता है और $3.4E-4932$ से $3.4E+4932$ मान तक की संख्या Store कर सकता है। इस तरह का Identifier भी हम निम्नानुसार Declare कर सकते हैं:

```
double lightMovementIn100Year;
```

Character

जब हमें Computer में "सी" Character set के किसी Character को Store करने के लिए Memory को Reserve करना होता है, तब हम Character प्रकार के Data Type का प्रयोग करके Identifier Create करते हैं। इस प्रकार का Identifier Create करने के लिए हमें "C" Language के **char** Keyword का प्रयोग करना होता है। इस प्रकार का Identifier मेमोरी में 1 Byte की Space Reserve करता है। char प्रकार के Identifiers में हम केवल एक ही Character Store कर सकते हैं। हम char प्रकार के Identifier में संख्या भी Store कर सकते हैं। इस Data Type को भी दो भागों में बांटा गया है:

signed char or char

Computer की Memory में हम कभी भी किसी Character को Store नहीं करते हैं। यदि हम किसी Character को Store भी करते हैं, तो वह Character किसी ना किसी अंक के रूप में ही Computer में Store होता है। Computer में हर Character का एक ASCII Code होता है।

यदि हम किसी Character को Computer की किसी Memory Location पर Store करते हैं, तो वास्तव में हम एक Integer मान को ही Computer की Memory में Store कर रहे होते हैं। लेकिन उस Memory Location पर Stored Character को Output में Display करने के तरीके पर निर्भर करता है, कि वह Character हमें एक Character के रूप में दिखाई देगा या फिर एक अंक के रूप में।

चूंकि **char** प्रकार का Data Type Memory में केवल एक Byte की ही Space लेता है, इसलिए जब हमें Computer में बहुत ही छोटी लेकिन चिन्ह वाली संख्या को Store करना होता है, तब इस Data Type के Identifier का प्रयोग कर सकते हैं। इस प्रकार के Identifier में हम -128 से 127 तक की संख्या Store कर सकते हैं। किसी Character प्रकार के Identifier को Declare करने के लिए हमें निम्न तरीके को Use करना होता है:

```
char studentAge;           or
signed char studentAge;
```

यदि हम इस Identifier में किसी Character को Store करना चाहें, तो Store किए जाने वाले Character को हमें Single Quote में लिखना होता है। यानी:

```
studentAge = '9';
```

इस Statement में हम Variable में 9 Store कर रहे हैं, लेकिन वास्तव में हम यहां पर अंक 9 Store नहीं कर रहे हैं, बल्कि अंक 9 की ASCII Value **57** या Character **9** Store कर रहे हैं। यदि हम इस Variable में अंक 9 Store करना चाहें, तो हमें ये Statement निम्नानुसार लिखना होगा:

```
studentAge = 9;
```

unsigned char

जब हमें Computer में ऐसे मान को Store करना होता है, जो कि बहुत छोटा तो होता है साथ ही कभी भी Minus में नहीं हो सकता है, तब हम इस के Identifier को Declare करते हैं।

उदाहरण के लिए किसी भी Student की Age Minus में नहीं हो सकती है, इसलिए Age को **signed** प्रकार का Declare करने की जरूरत नहीं है, बल्कि Age को **unsigned** प्रकार का Declare किया जाना चाहिए साथ ही चूंकि किसी भी व्यक्ति की Age सामान्यतया 255 साल से अधिक नहीं हो सकती है, इसलिए Age Integer होने के बावजूद Age को **int** प्रकार का Declare करने की जरूरत नहीं है। क्योंकि **unsigned char** प्रकार का Identifier 0 से 255 तक के मान की संख्या को Store कर सकता है। **unsigned char** प्रकार का Identifier Create करने के लिए हम निम्नानुसार Statement लिख सकते हैं:

```
unsigned char studentAge = 20;
```

अलग-अलग प्रकार के Data Type में जो मेमोरी Space बताया गया है, उसका अर्थ यही है कि ज्यादा मेमोरी Space लेने वाले Identifier में बड़ी संख्या व कम Storage Space लेने वाले Identifier में छोटी संख्या को Store किया जा सकता है।

Data Types Modifiers

ये मानक डाटा टाइप की साईज बदल देते हैं यानी ये डाटा टाइप के आकार में परिवर्तन कर देते हैं। ये कुल चार प्रकार के होते हैं:

- ◆ **Signed**
- ◆ **Unsigned**
- ◆ **Short**
- ◆ **Long**

हमने ऊपर इनके प्रयोग से देखा है कि कैसे **double** प्रकार का Identifier Memory में 8 Byte का Space लेता है और **double** के साथ **short** Modifier Use करने से वह Identifier 10 Byte की जगह Reserve कर लेता है।

विभिन्न प्रकार के Identifier Memory में कितनी Space Reserve करते हैं, इस बात की पूरी जानकारी "C" Language की Library में उपस्थित **limits.h** व **float.h** नाम की Header Files में दी गई है। लेकिन इन Header Files से इन जानकारियों को Screen पर Display करवाने के लिए पहले हमें "C" Language के Output Function **printf()** व इसमें प्रयोग किए जाने वाले विभिन्न प्रकार के Control Strings को ठीक से समझना होगा।

Exercise:

- 1 Data किसे कहते हैं ? Real World में मूल रूप से Data कितने प्रकार के होते हैं?
- 2 “C” Language में विभिन्न प्रकार के Data को Represent करने के लिए Data को किन दो भागों में बांटा गया है? इन दोनों भागों द्वारा किन-किन Data Types को Represent किया जाता है?
- 3 किस प्रकार के Real World Data को “C” Language के किस Data Type के Identifier में Store किया जाना चाहिए, इस बात का Decision किस प्रकार से लिया जाता है ?
- 4 निम्न Data Types के आपसी अन्तर को समझाईए:

A	int	float
B	short	long
C	signed	unsigned
- 5 Data Type modifiers से आप क्या समझते हैं ? इनका प्रयोग क्यों किया जाता है ? समझाईए।
- 6 सभी प्रकार के Data types का एक-एक उचित **Variable** व **Constant** Declare कीजिए।
- 7 निम्न Declarations में के हर Identifier के मान को यदि Output में Display किया जाए, तो इन Identifiers में Store किए गए सभी मानों को ज्यों का त्यों Output में प्राप्त करने के लिए हमें इन में से किन-किन Declarations किस तरह से Modify करना होगा:

A	const int age;	B	signed speed = 125.50
C	short velocity = 1.2e+4	D	long lightSpeed = 3.8e+10
E	unsigned float x = 1.5	F	unsigned long double p=1.5
G	const signed char ;	H	char x = 254;

Control String

जिस तरह से हम “C” Language में विभिन्न प्रकार के Data को Store करने के लिए अलग-अलग Keywords का प्रयोग करके अलग-अलग **Limit** की Memory Location को Reserve किया जाता है, ठीक इसी तरह से अलग-अलग प्रकार के मानों को Access करने के लिए भी हमें अलग-अलग तरह के Control Strings का प्रयोग करना होता है। Control String कुछ ऐसे Characters होते हैं, जिन्हें % के साथ Use किया जाता है।

उदाहरण के लिए यदि हम किसी Integer संख्या को Memory में Store करते हैं, तो उस Integer संख्या को Screen पर Display करने के लिए हमें %d Control String का प्रयोग करना होता है। इसी तरह से यदि हम Character प्रकार के किसी Data को Screen पर Print करना चाहें, तो हमें %c Control String का प्रयोग करना होता है। विभिन्न प्रकार के Data Type के Data को Screen पर Display करने के लिए printf() Function के साथ Use किए जाने वाले Control String को हम निम्न सारणी द्वारा समझ सकते हैं:

%d	Integer Data Type के मान को Display करने के लिए।
%c	Character Data Type के मान को Display करने के लिए।
%f	Real Number Data Type के मान को Display करने के लिए।
%g	Floating Point Real Data Type के मान को दसमलव के बाद केवल एक Digit तक के Round Off Form में Display करने के लिए
%i	Signed Decimal Integer Data Type के मान को Display करने के लिए।
%u	Unsigned Decimal Integer Data Type के मान को Display करने के लिए।
%o	Octal Integer Data Type के मान को Display करने के लिए।
%s	String Data Type के मान को Display करने के लिए।

- %x** Hexadecimal Data Type के मान को Display करने के लिए।
- %e** Real Number Data Type के मान को Display करने के लिए, जबकि संख्या का मान घातांक रूप में हो

विभिन्न प्रकार के Data Type के मानों को Access करने के लिए हमें विभिन्न प्रकार के Control Strings का प्रयोग करना पड़ता है। किस प्रकार के Identifier को Access करने के लिए किस Control String को Use करना चाहिए, इस बात की जानकारी निम्न सारणी द्वारा प्राप्त की जा सकती है:

Data Type	Control String
signed char unsigned char	%c
short signed int signed int	%d
short unsigned int unsigned int	%u
long signed int	%ld
long unsigned int	%lu
float	%f / %e
double	%lf / %le
long double	%Lf / %Le

float, **double** या **long double** Type के मानों को यदि Normal Form में Display करना हो, तो क्रमशः **%f**, **%lf** व **%Lf** Control Strings का प्रयोग करते हैं, जबकि यदि इनके मानों को घातांक रूप में Display करना हो, तो इनके लिए हमें क्रमशः **%e**, **%le** व **%Le** Control Strings का प्रयोग करना होता है।

printf() Function का प्रयोग हम किसी भी प्रकार के Numerical या Alphanumerical मान को Monitor पर Display करने के लिए करते हैं। इस Function में हमें जो भी Message Screen पर Display करना होता है, उस Message को हम String के रूप में Double Quotes के बीच में लिखते हैं। Double Quotes के बीच में लिखा गया Message ज्यों का त्यों Screen पर Display हो जाता है। उदाहरण के लिए यदि हमें Screen पर **“Hello World”** Print करना हो, तो हमें **printf()** Function में इस Message को निम्नानुसार लिखना होता है:

```
printf("Hello World");
```

इस Statement का Output हमें निम्नानुसार प्राप्त होता है:

```
Hello World
```

यदि हम इसी Statement को निम्नानुसार लिखते हैं:

```
printf("    Hello                World");
```

जो इस Statement का Output भी हमें निम्नानुसार प्राप्त होता है:

```
    Hello                World
```

यानी **Printf()** Statement में हम String को जिस Format में लिखते हैं, Output में हमें वह String उसी Format में दिखाई देता है। लेकिन विभिन्न प्रकार की Calculations के बाद प्राप्त होने वाले Result को Display करने के लिए भी हमें **printf()** Function का ही प्रयोग करना होता है।

इस स्थिति में हमें Display किए जाने वाले Data के Data Type के आधार पर किसी ना किसी Control String का प्रयोग करना पड़ता है।

जब हम Control String का प्रयोग करके किसी Calculated मान को Screen पर Display करना चाहते हैं, तब हमें हमेशा Data के Source व Data के Target दोनों को printf() Function में Specify करना जरूरी होता है, जहां Source वह मान होता है, जिसे Monitor पर Display करना होता है, जबकि Target वह स्थान होता है, जहां पर Data के मान को Display करना है। Target के स्थान पर Display किए जाने वाले Data के Data Type के Control String को Specify करना होता है। इस तरह से यदि हम printf() Function का पूर्ण Syntax देखें तो वह Syntax निम्नानुसार होता है:

Syntax:

```
printf("Message cntrlStr1 Message cntrlStr2...Message cntrlStrN",  
value/Identifir1, value/Identifir2 ... value/IdentifirN)
```

इस Syntax में **Message** के स्थान पर हम उस String को लिखते हैं, जिसे ज्यों का त्यों Screen पर Display करना होता है, जबकि **cntrlStr** के स्थान पर हम उस Control String का प्रयोग करते हैं, जो **value/Identifier** में Stored Data Type के मान को Display करने में सक्षम होता है।

cntrlStr व value/Identifier दोनों एक दूसरे के समानान्तर होते हैं। यानी cntrlStr1 के स्थान पर value/Ddentifier1 का मान ही Display होगा, cntrlStr2 के स्थान पर value/Ddentifier2 का मान ही Display होगा और cntrlStrN के स्थान पर value/IdentifierN का मान ही Display होगा। इनके क्रम में किसी प्रकार का कोई परिवर्तन नहीं किया जा सकता है।

यानी यदि हम चाहें कि **cntrlStr1** के स्थान पर **value/Identifier2** का मान Display हो, तो बिना printf() Statement में Change किए हुए हम ऐसा नहीं कर सकते हैं। यदि हमें cntrlStr1 के स्थान पर value/Identifier2 का मान Display करना हो, तो हमें **printf()** Syntax निम्नानुसार लिखना होगा:

Syntax:

```
printf("Message cntrlStr1 Message cntrlStr2...Message cntrlStrN",  
value/Identifir2, value/Identifier1 ... value/IdentifierN)
```

निम्न Program द्वारा हम विभिन्न प्रकार के Control Strings को Use करने की प्रक्रिया को समझ सकते हैं:

Program:

```
#include <stdio.h>  
#include <conio.h>  
main()  
{  
    printf("\n Integer = %d", 10);  
    printf("\n Character = %c", 'X');  
    printf("\n Float = %f", 13.2);  
    printf("\n Double = %e", 12365.599999);  
    printf("\n Double = %g", 12365.599999);  
    printf("\n String = %s", "Hello World");  
    getch();  
}
```

Output:

```
Integer = 10
Character = X
Float = 13.200000
Double = 1.236560e+04
Double = 12365.6
String = Hello World
```

इस Program में Monitor पर String को Display करने की सुविधा प्राप्त करने के लिए हमने **stdio.h** नाम की Header File को अपने Source Program में Include किया है, क्योंकि printf() Function को इसी Header File में Define किया गया है, जो कि Monitor पर Output को Display करने का काम करता है।

getch() Function को **conio.h** नाम की Header File में Define किया गया है, इसलिए हमने getch() की सुविधा को प्राप्त करने के लिए इस Header File को भी अपने Program में Include किया है।

फिर हमने main() Function बनाया है, क्योंकि यही वह Function होता है, जिसमें Computer को दिए जाने वाले विभिन्न Instructions को लिखा जाता है।

हम देख सकते हैं कि सभी printf() Functions में हमने “\n” को Use किया है। इस Character को **Backslash Character Constant** कहते हैं। ये Constant Computer को हर Message Print करने के पहले एक **New Line** लेने का Instruction देता है, ताकि Display होने वाला हर Statement एक नई Line में Display हो। यदि हम printf() Function में इस Character Constant का प्रयोग ना करें, तो इस Program का Output हमें एक ही Line में प्राप्त होगा।

इस Program में हम देख सकते हैं कि हर Statement में जिस स्थान पर Control String का प्रयोग किया गया है, Output में उसी स्थान पर वह मान Display हो रहा है, जो मान Source के रूप में Specify किया गया है।

इस Program में हमने हर मान को बिना Memory Allocate किए ही Directly Screen पर Display करने के लिए भेज दिया है। यदि हम चाहें तो सभी प्रकार के मानों को Display करने से पहले उन्हें Memory प्रदान कर सकते हैं और हर Memory को Refer करने के लिए Identifier Set कर सकते हैं। इस प्रक्रिया को Use करते हुए हम पिछले Program को ही निम्नानुसार Modify कर सकते हैं:

Program:

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

main()
{
    int Integer = 10;
    char Character = 'X';
    float Float = 13.2;
    double Double = 12365.599999;
    char String[] = "Hello World";
```

```

clrscr();

printf("\n Integer = %d", Integer);
printf("\n Character = %c", Character);
printf("\n Float = %f", Float);
printf("\n Double = %e", Double);
printf("\n Double = %g", Double);
printf("\n String = %s", String);
getch();
}

```

इस Program का Output भी हमें वही प्राप्त होता है, जो पिछले Program का प्राप्त हुआ है। लेकिन इस Program में विभिन्न प्रकार के मान Computer की Memory में प्रत्यक्ष रूप से विद्यमान हैं, जिन्हें किसी दूसरी प्रक्रिया के लिए भी Use किया जा सकता है।

“C” Language में Characters के समूह यानी String को Memory में Store करने के लिए किसी प्रकार का कोई Primary Data Type नहीं है, बल्कि String को Computer की Memory में Store करने के लिए हमें Character प्रकार का एक **One-Dimensional Array** Create करना होता है।

Preprocessor Directive

कई बार हमें ऐसी जरूरत होती है जिसमें हम चाहते हैं कि हमारा Source Program Compiler पर Compile होने के लिए Processor पर जाने से पहले कुछ काम करे। इस प्रकार के कामों को परिभाषित करने के लिए हम Preprocessor Directives का प्रयोग करते हैं। Preprocessor Directives की शुरुआत हमेशा # से होती है और इन्हें हमेशा Header Files को Include करने वाले Statement के Just नीचे लिखा जाता है। Preprocessor Directives को समझने के लिए हम एक Program देखते हैं, जिसमें हम Output में “Hello World” Print करना चाहते हैं। ये Program निम्नानुसार है:

Program:

```

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

#define START      main() {
#define PRINT      printf("Hello World");
#define PAUSE      getch();
#define END        }

START              /* Start the program. */
PRINT              /* Display message on the screen.*/
PAUSE              /* Pause the screen to display output. */
END                /* Terminate the program. */

```

Output:

Hello World

इस Program को Compile करने पर भी हमें वही Output प्राप्त होता है, जो Output हमें पिछले अध्याय में प्राप्त हुआ था। ऐसा इसलिए होता है, क्योंकि जब भी हम इस Program को Compile

करते हैं, इस Program में Define किए गए सभी Preprocessor Directives Program के Processor पर Compile होने के लिए जाने से पहले Expand होकर मूल Codes में Convert हो जाते हैं। जब सभी Directives Expand हो जाते हैं, तब ये Program निम्नानुसार Normal Form में आ जाता है:

Program:

```
#include <stdio.h>
#include <conio.h>
main() {
    printf("Hello World");
    getch();
}
```

/* START */
/* PRINT */
/* PAUSE */
/* END */

Compiler अब इस Normal Form में Converted Program को Processor पर Compile होने के लिए भेजता है। चूंकि **#define** के साथ Use किए जाने वाले Directives Program के Processor पर Compile होने के लिए जाने से पहले Expand होते हैं, इसलिए इन Directives को **Preprocessor Directives** कहा जाता है। विभिन्न प्रकार के Preprocessor Directives के बारे में हम आगे विस्तार से जानेंगे।

Preprocessor Directives का प्रयोग Header Files को Develop करते समय ही सबसे ज्यादा किया जाता है। "C" Language की Library में विभिन्न प्रकार के तकनीकी मानों को सरल रूप में Represent करने के लिए उन्हें Preprocessor Directives का प्रयोग करके एक सरल नाम दे दिया जाता है, ताकि इन मानों को सरलता से याद रखा जा सके व Use किया जा सके।

Preprocessor Directives का प्रयोग करके विभिन्न प्रकार के बड़े-बड़े तकनीकी नामों व मानों को सरल व छोटे Identifier के रूप में परिभाषित किया जा सकता है। एक बार किसी मान का कोई नाम दे देने के बाद हम उस मान को उसके नाम से Refer कर सकते हैं।

उदाहरण के लिए मानलो कि हमें किसी Program में बार-बार PI के मान **3.142857142857142** की जरूरत पड़ती है। अब इतने बड़े मान को बार-बार विभिन्न Statements में बिना गलती के लिखना, नामुमकिन है। किसी ना किसी Statement में इसको Type करने में Mistake हो ही जाएगी।

इस स्थिति में हम एक Preprocessor Directive का प्रयोग करके इस मान को एक नाम प्रदान कर सकते हैं। इस मान को एक नाम प्रदान कर देने के बाद हमें जिस किसी भी स्थान पर Calculation के लिए इस मान की जरूरत हो, हम उस नाम को Use कर लेते हैं।

जब Program को Compile करते हैं, तब Program Compile होने से पहले उन सभी स्थानों पर, जहां पर Preprocessor का प्रयोग किया गया है, Preprocessor को उसके मान से Replace कर देता है। इस तरह से Program में Typing की वजह से होने वाली गलतियों से बचा जा सकता है। इस प्रक्रिया को निम्न Program में Implement किया गया है।

Program:

```
#include <stdio.h>
#include <conio.h>
#define PI 3.142857142857142

main()
```

```
{  
    printf("Value of PI is = %.15e", PI);  
    getch();  
}
```

जब इस Program को Run किया जाता है, तब Output में **%2.15e** Control String के स्थान पर मान **3.142857142857142** को Represent करने वाले Identifier PI के स्थान पर ये मान Expand हो जाता है और Screen पर घातांक रूप में Display हो जाता है।

चूंकि Float प्रकार का Control String दसमलव के बाद केवल 6 अंकों तक की संख्या को ही Display कर सकता है, इसलिए हमने इस Program के printf() Function में **%.15e** Control String का प्रयोग किया है। ये Control String Compiler को ये Instruction देता है, कि Screen पर Display किए जाने वाले मान में दसमलव के बाद कुल 15 Digits Display होने चाहिए। ये Program जब Compile किया जाता है, तब Compile होने से पहले निम्नानुसार Form में Convert हो जाता है:

Program:

```
#include <stdio.h>  
#include <conio.h>  
  
main()  
{  
    printf("Value of PI is = %.15e", 3.142857142857142);  
    getch();  
}
```

जब हम अपनी जरूरत के आधार पर विभिन्न प्रकार के Identifier Declare करते हैं, तब किस प्रकार का Identifier Memory में Minimum व Maximum कितने मान तक की संख्या को Hold कर सकता है, इस बात की जानकारी "C" Language के साथ मिलने वाली **"limits.h"** नाम की Header File में दी गई है।

यदि हम चाहें तो इस Header File को जो कि ...TC\Include नाम के Folder में होती है, Open करके विभिन्न Data Types द्वारा Store की जा सकने वाली Minimum व Maximum Range का पता लगा सकते हैं। इस Header File में विभिन्न Data Type द्वारा प्रदान की जाने वाली Minimum व Maximum Range को कुछ Preprocessor Directives के रूप में Define किया गया है।

इसलिए यदि हम Header File को Open करके ना देखना चाहें, तो निम्नानुसार एक Program बना कर भी हम विभिन्न Data Types द्वारा प्रदान की जाने वाली Minimum व Maximum Range का पता लगा सकते हैं।

चूंकि इन विभिन्न प्रकार के Directives को **"limits.h"** नाम की Header File में Define किया गया है, इसलिए इन Directives को Access करने के लिए हमें **"limits.h"** नाम की Header File को अपने Program में Include करना जरूरी होता है।

Program:

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

```

#include <float.h>
#define _ printf("\n Minimum
#define __ printf("\n Maximum

void main(){
    _ short|short int|signed short|signed short int : %d ", SHRT_MIN );
    __ short|short int|signed short|signed short int : %d ", SHRT_MAX );
    _ unsigned short|unsigned short int : %u ", 0 );
    __ unsigned short|unsigned short int : %u ", USHRT_MAX );
    _ int|signed int : %d ", INT_MIN );
    __ int|signed int : %d ", INT_MAX );
    _ unsigned int : %u ", 0 );
    __ unsigned int : %u ", UINT_MAX );
    _ long|long int|signed long|signed long int : %ld ", LONG_MIN );
    __ long|long int|signed long|signed long int : %ld ", LONG_MAX );
    _ unsigned long|unsigned long int : %lu ", 0 );
    __ unsigned long|unsigned long int : %lu ", ULONG_MAX );
    _ float : %e", FLT_MIN);
    __ float : %e", FLT_MAX);
    _ double : %e", DBL_MIN);
    __ double : %e", DBL_MAX);
    _ long double : %Le", LDBL_MIN);
    __ long double : %Le", LDBL_MAX);
}

```

ये Program देखने में बहुत अजीब लग सकता है, लेकिन इस Program में हमने Underscore व Double Underscore Symbol से printf() Function के कुछ Part को Directive के रूप में परिभाषित कर लिया है। जब इस Program को Compile करते हैं, तब Program Compile होने से पहले Underscore (_) Symbol के स्थान पर “printf(“\n Minimum” String को व Double Underscore Symbol (__) के स्थान पर “printf(“\n Maximum” String Replace कर देता है। Preprocess होने के बाद जब Program Compile होकर Run होता है, तब हमें इस Program का Output निम्नानुसार प्राप्त होता है:

Output:

MinimumV short short int signed short signed short int	: -32768
MaximumV short short int signed short signed short int	: 32767
MinimumV unsigned short unsigned short int	: 0
MaximumV unsigned short unsigned short int	: 65535
MinimumV int signed int	: -2147483648
MaximumV int signed int	: 2147483647
MinimumV unsigned int	: 0
MaximumV unsigned int	: 4294967295
MinimumV long long int signed long signed long int	: -2147483648
MaximumV long long int signed long signed long int	: 2147483647
MinimumV unsigned long unsigned long int	: 0
MaximumV unsigned long unsigned long int	: 4294967295
MinimumV float	: 1.175494e-38
MaximumV float	: 3.402823e+38
MinimumV double	: 2.225074e-308
MaximumV double	: 1.797693e+308

MinimumV long double	: 3.362103e-4932
MaximumV long double	: 1.189731e+4932

limits.h नाम की Header File में विभिन्न प्रकार के Integers से सम्बंधित Range की जानकारी होती है, उसी तरह से Float से सम्बंधित विभिन्न प्रकार के Range की जानकारी के लिए हम "float.h" नाम की Header File को Open करके देख सकते हैं। इसीलिए हमने हमारे Program में Float व Double से सम्बंधित Range की जानकारी के लिए "float.h" नाम की Header File को भी अपने Program में Include किया है।

यदि हम ये जानना चाहें कि विभिन्न प्रकार के Data Type के Identifiers Memory में कितने Bytes की Space Reserve करते हैं, तो इस बात का पता लगाने के लिए हम **sizeof()** Operator का प्रयोग कर सकते हैं। ये Operator Argument के रूप में उस Identifier या Data Type को लेता है, जिसकी Size को हम जानना चाहते हैं और हमें उस Data Type या Identifier की Size Return करता है। यानी इस Operator के Bracket के बीच में हम जिस Identifier या Data Type को लिख देते हैं, हमें उसी Data Type की Size का पता चल जाता है।

सामान्यतया Integer Data Type के अलावा सभी Data Types सभी प्रकार के Computers में समान Memory Occupy करते हैं, जबकि **Integer**, Memory में Compiler के Register की Size के बराबर Space Reserve करता है।

यदि हम 16-Bit Compiler में 16-Bit Processor पर Program Develop करते या Run करते हैं, तो Integer 16-Bit System में 2-Bytes का होता है जबकि 32-Bit System में Integer की Size 4-Bytes होती है। हम जिस Compiler को Use कर रहे हैं, उस Compiler द्वारा विभिन्न प्रकार के Basic Data Type द्वारा Occupy की जा रही Memory का पता हम निम्न Program द्वारा लगा सकते हैं:

Program

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

void main()
{
    printf("char      : %d Bytes\n", sizeof(char));
    printf("short     : %d Bytes\n", sizeof(short));
    printf("int       : %d Bytes\n", sizeof(int));
    printf("long      : %d Bytes\n\n", sizeof(long));

    printf("signed char : %d Bytes\n", sizeof(signed char));
    printf("signed short : %d Bytes\n", sizeof(signed short));
    printf("signed int  : %d Bytes\n", sizeof(signed int));
    printf("signed long : %d Bytes\n\n", sizeof(signed long));

    printf("unsigned char : %d Bytes\n", sizeof(unsigned char));
    printf("unsigned short: %d Bytes\n", sizeof(unsigned short));
    printf("unsigned int  : %d Bytes\n", sizeof(unsigned int));
    printf("unsigned long : %d Bytes\n\n", sizeof(unsigned long));

    printf("float      : %d Bytes\n", sizeof(float));
    printf("double     : %d Bytes\n", sizeof(double));
}
```

```
printf("long double : %d Bytes\n", sizeof(long double));

getch();
}
```

Output:

16 - Bit Compiler's Output		32 - Bit Compiler's Output	
char	: 1 Bytes	char	: 1 Bytes
short	: 2 Bytes	short	: 2 Bytes
int	: 2 Bytes	int	: 4 Bytes
long	: 4 Bytes	long	: 4 Bytes
signed char	: 1 Bytes	signed char	: 1 Bytes
signed short	: 2 Bytes	signed short	: 2 Bytes
signed int	: 2 Bytes	signed int	: 4 Bytes
signed long	: 4 Bytes	signed long	: 4 Bytes
unsigned char	: 1 Bytes	unsigned char	: 1 Bytes
unsigned short	: 2 Bytes	unsigned short	: 2 Bytes
unsigned int	: 2 Bytes	unsigned int	: 4 Bytes
unsigned long	: 4 Bytes	unsigned long	: 4 Bytes
float	: 4 Bytes	float	: 4 Bytes
double	: 8 Bytes	double	: 8 Bytes
long double	: 10 Bytes	long double	: 10 Bytes

Exercise:

- 1 Control Strings से आप क्या समझते हैं? विभिन्न प्रकार के Control Strings को समझाते हुए इसे Use करने के तरीके का एक Program द्वारा वर्णन कीजिए।
- 2 Preprocessor Directives किसे कहते हैं? ये किस प्रकार से काम करता है और एक "C" Program में किस प्रकार से Use किया जा सकता है? एक उदाहरण देकर समझाईए।
- 3 "C" Language में Supported विभिन्न Data Types की Range Limit तथा हर Data Type द्वारा Reserve की जाने वाली Memory Size को एक Program द्वारा Screen पर Display कीजिए। साथ ही Program के Flow को भी समझाईए।

Literal

जब हम Computer में किसी Program को Develop करते हैं, तब Program में हमें कई ऐसे Data को भी Store व Access करना होता है, जिनका मान हमेशा स्थिर रहता है। इस प्रकार के Data को **Literal** या **Constant** कहते हैं। उदाहरण के लिए

- 1 एक Week में हमेशा 7 दिन होते हैं।
- 2 PI का मान हमेशा 22/7 होता है।
- 3 एक साल में हमेशा 12 महीने होते हैं।
- 4 एक दिन में हमेशा 24 Hours होते हैं।
- 5 एक Hour में हमेशा 60 मिनट होते हैं।

प्रोग्राम के Execution के दौरान Literals के मान में कोई परिवर्तन नहीं होता है। इनका मान सम्पूर्ण प्रोग्राम में स्थिर रहता है। चूंकि Literals हमेशा स्थिर Data को Represent करते हैं, इसलिए इन

मानों को Store करने के लिए Memory में Space Reserve करते समय ही इन मानों को Memory में Store कर दिया जाता है और ऐसी व्यवस्था कर दी जाती है, ताकि इनका मान पूरे Program में किसी भी स्थिति में Change ना किया जा सके। “C” Language में Literals या Constants को तीन भागों में बांटा गया है:

Integer Constant

इन्हें पूर्णांक संख्याएं भी कहते हैं, क्योंकि इनमें दशमलव वाली संख्याएं नहीं होती हैं। इस प्रकार के Constant में +/- चिन्ह हो सकते हैं। जिस अंक पर कोई चिन्ह न हो वह संख्या Positive होती है। जैसे 124, 3223, 545, 23 आदि Positive Integer Constant के उदाहरण हैं। Programming के दौरान बड़ी संख्याओं को Represent करने के लिए संख्याओं के बीच कोमा का प्रयोग नहीं किया जाता है। जैसे 1233,33,000 एक गलत स्थिरांक है। Integer Constants को भी मुख्यतः तीन भागों में बांटा जा सकता है:

Decimal Constant

जब हम Computer में किसी संख्या को 0 से 9 तक की Digits का प्रयोग करके Represent करते हैं, तब हम इस प्रकार के Constant को Decimal Integer Constant कहते हैं। इस तरीके को Number की Decimal Form कहा जाता है। हम हमारे दैनिक जीवन में संख्याओं को इसी रूप में उपयोग में लेते हैं। जब हम किसी Identifier में इस प्रकार के Literal को Assign करना चाहते हैं, तब हमें निम्नानुसार Syntax लिखना होता है:

```
const int speed = 120;
```

const Keyword का प्रयोग इसलिए किया जाता है, क्योंकि **const** Keyword का प्रयोग करने पर Create होने वाला Identifier **Constant Identifier** बन जाता है, जबकि **const** का प्रयोग ना करने पर बनने वाला Identifier **Variable Identifier** होता है। इस Statement में 120 एक **Decimal Literal** है।

Octal Constant

जब हम Computer में किसी संख्या को 0 से 7 तक की Digits का प्रयोग करके Represent करते हैं, तब हम इस प्रकार के Constant को Octal Integer Constant कहते हैं। इस तरीके को Number की Octal Form कहा जाता है और इस तरीके का प्रयोग केवल Electronic Devices जैसे कि Calculator, VCD Player, DVD Player, Remote Control आदि में Number को Represent करने के लिए किया जाता है।

जब हम किसी Number को इस तरीके का प्रयोग करके Represent करते हैं, तब इस मान के Number से पहले English Alphabet के एक Character **0** का प्रयोग किया जाता है, जो बताता है कि Represent होने वाली संख्या Octal Form में है।

हम हमारे दैनिक जीवन में इस तरीके का प्रयोग करके किसी Number को Represent नहीं करते हैं। जब हम किसी Identifier में इस प्रकार के Literal को Assign करना चाहते हैं, तब हमें निम्नानुसार Syntax लिखना होता है:

```
const int speed = 0120;
```

चूंकि जब हम किसी मान को Octal Value के रूप में Store या Access करना चाहते हैं, तब हमें उस मान के आगे उपसर्ग के रूप में **0 (Zero) Add** करना जरूरी होता है। इसीलिए हमने मान 120 से पहले Prefix के रूप में **0 (Zero)** का प्रयोग किया है। इस Statement में **0120** एक **Octal Literal** है।

Hexadecimal

जब हम Computer में किसी संख्या को 0 से 9 तक की Digits English के a/A, b/B, c/C, d/D, e/E या f/F Characters का प्रयोग करके Represent करते हैं, तब हम इस प्रकार के Constant को Hexadecimal Integer Constant कहते हैं।

इस तरीके को Number की Hexadecimal Form कहा जाता है और इस तरीके का प्रयोग Computer जैसी बड़ी Digital Electronic Devices में संख्याओं को Represent करने के लिए किया जाता है।

जब हम किसी Number को इस तरीके का प्रयोग करके Represent करते हैं, तब इस मान के Number से पहले **0X/0x** का प्रयोग किया जाता है, जो बताता है कि Represent होने वाली संख्या Hexadecimal Form में है।

हम हमारे दैनिक जीवन में इस तरीके का प्रयोग करके भी किसी Number को Represent नहीं करते हैं। जब हम किसी Identifier में इस प्रकार के Literal को Assign करना चाहते हैं, तब हमें निम्नानुसार Syntax लिखना होता है:

```
const int speed = 0x120;  
or  
const int speed = 0X120;
```

चूंकि जब हम किसी मान को Hexadecimal Value के रूप में Store या Access करना चाहते हैं, तब हमें उस मान के आगे उपसर्ग के रूप में **0x (Zero with x/X)** Add करना जरूरी होता है। इसीलिए हमने मान 120 से पहले Prefix के रूप में **0x (Zero with x/X)** का प्रयोग किया है। इस Statement में **0120** एक **Hexadecimal Literal** है।

Rules for Representing Integer Constants in a PROGRAM

किसी Program में जब भी हम किसी Integer Constant मान को Represent करते हैं, तब हमें कुछ नियमों का ध्यान रखना होता है। किसी Integer Constant को Represent करने के नियम निम्नानुसार हैं:

- किसी भी Integer Constant में कम से कम एक Digit को होना जरूरी होता है।
- किसी Integer Constant में कोई दसमलव नहीं होता है।
- Integer Constant **Positive** या **Negative** किसी भी प्रकार का हो सकता है।
- यदि किसी Integer Constant के साथ जब किसी चिन्ह का प्रयोग नहीं किया गया होता है, तब By Default वह Integer एक Positive Integer Constant होता है।
- किसी Integer Constant में अंकों को अलग करने के लिए **Blank Space** या **Comma** का प्रयोग नहीं किया जा सकता है।

इन नियमों को ध्यान में रखते हुए ही हमें हमारे Program में किसी Integer Constant को Represent करना होता है। इनमें से किसी भी नियम को Avoid करने पर “C” का Compiler **Compile Time Error** Generate करता है।

जब किसी Program को Compile करते समय Source Code में की गई किसी Typing Mistake के कारण कोई Error Generate होती है, तो इस Error को **Compile Time Error** कहते हैं।

हम जिस किसी भी रूप में जो भी स्थिर Integer मान Represent करते हैं, वह मान **Integer Literal** या **Integer Constant** कहलाता है। उदाहरण के लिए आगे दिए जा रहे सभी मान एक Integer Constant मान हैं:

Decimal Form	Octal Form	Hexadecimal Form
123	0123	0x124
+2568	+02568	0x5698
-7812	-01235	0x-4589
-5698	-124589	0x-7895

Floating Point Constant

जब हमें Computer में ऐसे Constant मान को Hold करना होता है, जिसमें दसमलव का प्रयोग होता है, तो इस प्रकार की संख्या को **Floating Point Constant** कहा जाता है। इस प्रकार की संख्या को **Real Number Constant** या वास्तविक स्थिरांक भी कहते हैं। “C” Language में इसे भी दो रूपों में Represent किया जाता है :

Fractional Form

जब किसी संख्या में स्थित दसमलव से पहले व दसमलव के बाद में दोनों ओर कम से कम एक अंक हो, तो इस प्रकार की संख्या को **Fractional Form Floating Point Constant** कहा जाता है। जैसे 12122.122, 11.22 आदि। इस प्रकार का Literal किसी Constant Identifier को Assign करने के लिए हमें निम्नानुसार Declaration करना होता है:

```
const float lightSpeed = 300000000.00;
```

इस Statement में 300000000.00 एक **Fractional Form Literal** है। चूंकि बड़ी संख्याओं को हमेशा सरलता से Use करने के लिए घातांक रूप में Convert करके Use किया जाता है। इसलिए इस Literal को भी हम Exponent Form में Convert करके Use कर सकते हैं।

Exponent Form

जब हम किसी संख्या को घातांक के रूप में Computer में Represent करते हैं, तो उस प्रकार की संख्या को **Exponent Form Floating Point Constant** कहा जाता है। ऐसी संख्या के हमेशा निम्नानुसार दो भाग होते हैं:

- 1) **Mantissa** व 2) **Exponent**

इस तरीके का प्रयोग करके बड़ी-बड़ी संख्याओं को घातांक के रूप में दर्शाया जाता है। जैसे 1200000000 को घातांक रूप में हम निम्नानुसार भी लिख सकते हैं। $1200000000 = 1.200000000 * 10^{10}$ जहां 1.2 Mantissa वाला भाग होगा व 10^{10} Exponent वाला भाग हो

जाएगा। किसी भी Fractional Form मान को Exponent Form में Convert करने के लिए निम्न सूत्र का प्रयोग कर सकते हैं:

Value	=	Mantissa	e/E	Exponent
1200000000	=	1.2	+E	10 = 1.2+E10

इस तरह से किसी भी संख्या को घातांक रूप प्राप्त किया जा सकता है। यदि घातांक धनात्मक हो तो +e या +E आता है अन्यथा -e या -E आता है। इस प्रकार का Literal किसी Constant Identifier को Assign करने के लिए हम निम्नानुसार Declaration कर सकते हैं:

```
const float lightSpeed = 3.0+E10;
```

इस Statement में 300000000.00 एक **Exponent Form Literal** है।

Rules for Representing Real Constants in a PROGRAM

किसी Program में जब भी हम किसी Real Constant मान को Represent करते हैं, तब हमें कुछ नियमों का ध्यान रखना होता है। किसी Real Constant को Represent करने के नियम निम्नानुसार हैं:

- किसी भी Real Constant में कम से कम एक Digit को होना जरूरी होता है।
- किसी Real Constant में हमेशा एक दसमलव होता है और दसमलव के बाद कम से कम एक Digit का होना जरूरी होता है।
- Real Constant भी **Positive** या **Negative** किसी प्रकार का हो सकता है।
- यदि किसी Real Constant के साथ जब किसी चिन्ह का प्रयोग नहीं किया गया होता है, तब By Default वह Real Constant एक Positive Real Constant होता है।
- किसी Real Constant में अंकों को अलग करने के लिए **Blank Space** या **Comma** का प्रयोग नहीं किया जा सकता है।

इन नियमों को ध्यान में रखते हुए ही हमें हमारे Program में किसी Integer Constant को Represent करना होता है। इनमें से किसी भी नियम को Avoid करने पर "C" का Compiler **Compile Time Error** Generate करता है।

हम जिस किसी भी रूप में जो भी स्थिर **Real** मान Represent करते हैं, वह मान **Real Literal** या **Real Constant** कहलाता है। उदाहरण के लिए आगे दिए जा रहे सभी मान Real Constant मान हैं:

Fractional Form	Exponential Form
123.32	12.3e+12
+2568.23	+2.568e+32
-7812.12	-123.5e5
-5698.21	-1245.89-e10

हम किसी Literal को सामान्य Form में Store करके उसे किसी भी दूसरे Form में भी Display करवा सकते हैं। उदाहरण के लिए यदि हम किसी Integer मान Decimal Form में Store करते हैं, तो उस मान को Display करवाते समय %d, %i, %o या %x Control Strings का प्रयोग करके Decimal, Octal या Hexadecimal Form में Display करवा सकते हैं।

इसी तरह से यदि हम किसी Floating Point Value को किसी Identifier में सामान्य रूप में Store करते हैं, तब भी हम उस मान को सामान्य दसमलव वाली संख्या व घातांक वाली संख्या दोनों ही रूपों में Output में Display करवा सकते हैं। इस पूरी प्रक्रिया को निम्न Program द्वारा समझा जा सकता है:

Program

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

main()
{
    const int Integer = 12345;
    const float Float = 12345.6789;

    printf("\n Integer in Decimal Form    = %d ", Integer);
    printf("\n Integer in Octal Form      = %o ", Integer);
    printf("\n Integer in Hexadecimal Form = %x ", Integer);
    putchar('\n');
    printf("\n Float in Fractional Form = %f ", Float);
    printf("\n Float in Exponential Form = %e ", Float);

    getch();
}
```

Output

```
Integer in Decimal Form    = 12345
Integer in Octal Form      = 30071
Integer in Hexadecimal Form = 3039

Float in Fractional Form = 12345.678711
Float in Exponential Form = 1.234568e+04
```

इस Program में हमने केवल एक ही Integer मान को तीन तरीकों से व एक ही Floating Point मान को दो तरीकों से Output में Display करने के लिए अलग-अलग **Control Strings** का प्रयोग किया है।

Output में हम देख सकते हैं कि Control String को बदल देने से हमें Output में प्राप्त होने वाला मान किसी दूसरे Form में दिखाई देने लगता है, जबकि हमें Actual Identifier के मान को Change करने की जरूरत नहीं होती है।

इस Discussion का सारांश ये है कि हम चाहे किसी भी Form में Calculation को Perform करें, उसे Output में Display करने पर हमें किस Form में Output चाहिए, इस बात को printf() Function में अलग-अलग Control String का प्रयोग करके तय किया जा सकता है।

इस Program में हमने **putchar()** नाम का एक नया Function Use किया है। इस Function को भी **stdio.h** नाम की Header File में ही Define किया गया है। इस Function में हम Argument के रूप में एक Character Pass करते हैं और ये Function उस Character को Screen या Output Device पर भेज देता है।

चूंकि हमने इस Function में Output Screen पर एक New Line प्राप्त करने के लिए ‘\n’ Backslash Character Constant को Argument के रूप में Pass किया है, इसलिए ये Function हमें Output में एक New Line Provide करता है। यदि हम Argument के रूप में किसी अन्य Character को इस Function में Pass करते, तो वह Character भी Output में ज्यों का त्यों Print हो जाता है।

चूंकि हमें इस Function में हमेशा एक ही Character को Argument के रूप में भेजना होता है, इसलिए इस Function में Argument के रूप में Pass किए जाने वाले Character को Single Quotes के बीच लिखकर भेजना होता है।

Character Constant

कई बार हमें कुछ ऐसे Data को Computer में Store करना होता है, जो एक या एक से अधिक Alphanumeric Character के होते हैं। इस प्रकार के Constant मान को *Character Constant* कहा जाता है। “C” Language में Character Constant भी तीन तरह के होते हैं:

Single Character Constant

जब कभी हमें Computer में ऐसे सवालों का जवाब Store करना होता है, जो केवल True/False या Yes/No के रूप में होते हैं, तब हम इस प्रकार के सवालों के जवाब को Represent करने के लिए एक Single Character का प्रयोग करते हैं। इस प्रकार के Constant को *Single Character Constant* कहा जाता है।

इसे हमेशा Single Quote द्वारा Represent करते हैं। उदाहरण के लिए मानलो कि हमें किसी Character Identifier में एक Character को Store करना है। इस काम को करने के लिए हमें निम्नानुसार Statement लिखना होगा:

```
char isTrue = 'y';
```

इस Statement में ‘y’ एक Character Literal है, जिसे Single Quote में Specify किया गया है।

String Constant

जब हमें Computer में कुछ Characters के एक समूह को Store करना होता है, जो कि एक स्थिर मान को Represent करता है, तो उस स्थिति में हम Alphanumerical Characters के एक समूह Computer में Store करते हैं। इस Characters के समूह को *String Constant* कहा जाता है।

String को हमेशा Double Quotes के बीच में लिखते हैं। इस प्रकार का Identifier Declare करने के लिए हमें निम्नानुसार एक **One-Dimensional Array** बनाना होता है, क्योंकि “C” Language में String Constant को Hold करने के लिए किसी प्रकार का कोई Standard Data Type नहीं है:

```
const char firstDayOfWeek [ ] = "MONDAY";  
const char firstMonthOfYear [ ] = "January";  
const char independenceDayOfIndia [ ] = "15-Aug-1947";
```

इन तीनों Statements में “MONDAY”, “January” व “15-Aug-1947” String Literals हैं।

Back slash Character Constant

जब हम कोई Program Develop करते हैं, तब उसका Output अच्छे Format में दिखाने के लिए हम कुछ विशेष प्रकार के Character Constants का प्रयोग कर सकते हैं, जिन्हें Back Slash के साथ उपयोग में लिया जाता है। इस प्रकार के Character Constants को *Backslash Character Constant* कहा जाता है। “C” Language में Support किए जाने वाले विभिन्न Backslash Character Constants निम्नानुसार हैं:

<code>\a'</code>	Bell	<code>\b'</code>	Back Slash
<code>\f'</code>	Form Feed	<code>\n'</code>	New Line
<code>\r'</code>	Carriage Return	<code>\t'</code>	Horizontal Tab
<code>\v'</code>	Vertical Tab	<code>\'</code>	Single Quote
<code>\"</code>	Double Quote	<code>\?'</code>	Question Mark
<code>\\'</code>	Back Slash	<code>\0'</code>	NULL Character

Rules for Representing Character Constants in a PROGRAM

किसी Program में जब भी हम किसी Character Constant मान को Represent करते हैं, तब हमें कुछ नियमों का ध्यान रखना होता है। किसी Character Constant को Represent करने के नियम निम्नानुसार हैं:

- किसी भी Character Constant में अधिकतम एक ही Character को Represent किया जा सकता है।
- किसी Character Constant में Represent किए जाने वाले Character को हमेशा एक Opening Single Quote के बीच ही लिखा जाता है। Character Constant के दोनों तरफ प्रयोग किया जाने वाला ये Quote हमेशा Opening Quote ही होना चाहिए।
- Character Constant के रूप में “C” Character set के किसी भी Character को Represent किया जा सकता है।

इन नियमों को ध्यान में रखते हुए ही हमें हमारे Program में किसी Character Constant को Represent करना होता है। इनमें से किसी भी नियम को Avoid करने पर “C” का Compiler **Compile Time Error** Generate करता है।

हम जिस किसी भी रूप में जो भी स्थिर **Character** मान Represent करते हैं, वह मान **Character Literal** या **Character Constant** कहलाता है। उदाहरण के लिए आगे दिए जा रहे सभी मान Character Constant मान हैं:

`\A', \2', \0', \t', \y', \n', \$', \#, \-', \='`

विभिन्न प्रकार के Backslash Character Constants का प्रयोग सामान्यतया printf() जैसे Output Functions में किया जाता है। इनका प्रयोग करने से हमें Output Screen पर दिखाई देने वाले Output को कुछ हद तक Format करने की सुविधा प्राप्त होती है।

विभिन्न प्रकार के Backslash Character Constants का प्रयोग विभिन्न प्रकार की स्थितियों में किया जाता है। उदाहरण के लिए यदि Output Screen पर New Line की जरूरत हो, तो हम '\n' Character Constant को Use करते हैं।

यदि हमें किसी Error को High Light करना हो, तो हम '\a' Character Constant का प्रयोग कर सकते हैं। इसी तरह से यदि हमें Output Screen पर Horizontal Tab की जरूरत हो, तो हम '\t' Character Constant को Use कर सकते हैं।

Exercise:

- वर्णन करते हुए Literal को परिभाषित कीजिए।
- Compile Time Error किसे कहते हैं और ये कब Generate होती है ?
- Integer Literals को कितने तरीकों से Represent किया जा सकता है ? सभी तरीकों से चार-चार उचित Literals को Represent कीजिए।
- किसी Program में Integer Constants को Represent करने के नियमों का वर्णन कीजिए।
- Floating Point Constant मानों को कितने तरीकों से Represent किया जा सकता है ? विभिन्न तरीकों से दो-दो उचित Literals Represent कीजिए।
- निम्न Floating Point Constants को Exponential Form में Convert कीजिए:
A 12536.369 B -4589.2
C 789.124587369 D 7889.2356
- निम्न Floating Point Constants को Fractional Form में Convert कीजिए:
A 1.2369e+1 B 8920000e-10
C 7.9E+9 D 2.356E-6
- किसी Real Constant को Program में Represent करने के नियमों का वर्णन कीजिए।
- 123.54 व 897 को इनके विभिन्न रूपों में Display करने का Program बनाईए।
- Character Constants कितने प्रकार के होते हैं ? इनका प्रयोग कब किया जाता है ?
- एक Program द्वारा विभिन्न प्रकार के Backslash Character Constants को समझाईए।
- Character Constant Represent करते समय किन नियमों को ध्यान में रखना होता है ?
- विभिन्न प्रकार के Character Constants के चार-चार उदाहरण दीजिए।

Punctuation

कुछ Special Symbols का प्रयोग प्रोग्राम में शब्दों व वाक्यों को अलग करने के लिए किया जाता है। इन्हें Punctuation या Separator कहते हैं। इनका मुख्य काम Program के एक Statement को दूसरे Statement व एक हिस्से को दूसरे हिस्से से अलग करने का होता है।

- [] Array की Size Define करने में उपयोग होता है।
- { } सभी Functions के Executables Code इन्ही कोष्ठकों के बीच लिखे जाते हैं।
- () ये चिन्ह बताता है कि Use हो रहा Statement एक Function है।
- ,
- ;
- :
- *
- # Preprocessor Directive है।

इन विभिन्न प्रकार के चिन्हों का प्रयोग हमें समय-समय पर जरूरत के आधार पर करना पड़ता है। उदाहरण के लिए जब भी हमें Header Files को Include करना होता है, हम # Preprocessor Directive का प्रयोग करते हैं। जब हम कोई नया Function Define करते हैं या किसी Function

को Call करते हैं, तब हमें Function के नाम के साथ () Braces का प्रयोग करना होता है, जैसाकि हम पिछले Programs में करते आ रहे हैं।

जब हम कोई Function Define करते हैं, तब इस कोष्ठक के बीच में Argument List को Specify किया जाता है और इसके बाद Semicolon का प्रयोग नहीं किया जाता है, जबकि किसी Function को Call करते समय हमें इस कोष्ठक में Argument Pass करना होता है और कोष्ठक के बाद Statement का अन्त दर्शाने के लिए Semicolon का प्रयोग करना जरूरी होता है।

ठीक इसी तरह से किसी भी Function के Body की शुरुआत व अन्त को Represent करने के लिए Opening व Closing Curly Braces { } का प्रयोग किया जाता है। इन Braces के बीच लिखे गए Statements के समूह को Block Statement भी कहा जाता है। Block Statement की विशेषता ये होती है, कि या तो Block के सभी Statement Execute होते हैं या फिर एक भी Statement Execute नहीं होता है।

Operators

किसी भी प्रोग्रामिंग भाषा में विभिन्न प्रकार के Results प्राप्त करने के लिए विभिन्न प्रकार के Mathematical व Logical Calculations करने पड़ते हैं। इन विभिन्न प्रकार के Mathematical व Logical Calculations को Perform करने के लिए कुछ Special Symbols का प्रयोग किया जाता है। ये Special Symbols कम्प्यूटर को विभिन्न प्रकार के Calculations करने के लिए निर्देशित करते हैं।

विभिन्न प्रकार के Calculations को Perform करने के लिए Computer को निर्देशित करने वाले चिन्हों को Operators कहा जाता है। साथ ही Data को Refer करने वाले जिन Identifiers के साथ ये प्रक्रिया करते हैं, उन Identifiers को इन Operators का Operand कहा जाता है। Operators दो तरह के होते हैं:

Unary Operator

कुछ Operators ऐसे होते हैं, जिन्हें कोई Operation Perform करने के लिए केवल एक Operand की जरूरत होती है। ऐसे Operator Unary Operator कहलाते हैं। जैसे Minus (-) एक Unary Operator है। जिस किसी भी संख्या के साथ ये चिन्ह लगा दिया जाता है, उस संख्या का मान बदल जाता है। जैसे 6 के साथ - चिन्ह लगा देने से संख्या -6 हो जाती है। "C" Language में Support किए गए Unary Operators निम्नानुसार हैं।

&	Address Operator
*	Indirection Operator
+	Unary Plus
-	Unary Minus
~	Bit wise Operator
++	Unary Increment Operator
--	Unary Decrement Operator
!	Logical Operator

Binary Operators

जिन Operators को काम करने के लिए दो Operands की जरूरत होती है, उन्हें **Binary Operators** कहते हैं। जैसे $2 + 3$ को जोड़ने के लिए Addition Operator (+) को दो Operands की जरूरत होती है, अतः Plus एक Binary Operator भी है।

“सी” Language में विभिन्न प्रकार के Operators को उनके काम के आधार पर निम्न Categories में बांटा गया है:

Arithmetic Operators

इनका उपयोग गणित के संख्यात्मक मानों की गणना करने के लिए किया जाता है। इन Operators की कुल संख्या पांच होती है, जो कि निम्नानुसार है:

```
// A = 10, B = 3, C = ?
```

```
//-----
```

1 Addition Operator (+)

ये Operator दो Operands को जोड़ कर उनका **योगफल** Return करता है। जैसे

```
C = A + B
```

```
C = 10 + 3
```

```
C = 13
```

2 Subtraction Operator (-)

ये Operator पहले Operand के मान में से दूसरे Operands के मान को घटाने पर प्राप्त होने वाले घटान या **घटाफल** को Return करता है। जैसे

```
C = A - B
```

```
C = 10 - 3
```

```
C = 7
```

3 Multiplication Operator (*)

ये Operator दोनों Operands के मानों को गुणा करके प्राप्त होने वाले **गुणनफल** को Return करता है। जैसे

```
C = A * B
```

```
C = 10 * 3
```

```
C = 30
```

4 Division Operator (/)

ये Operator पहले Operands के मान में दूसरे Operand के मान का भाग देकर प्राप्त होने वाले **भागफल** को Return करता है। जैसे

```
C = A / B
```

```
C = 10 / 3
```

```
C = 3
```

5 Modules OR Remainder Operator (%)

ये Operator पहले Operands के मान में दूसरे Operand के मान का भाग देकर प्राप्त होने वाले **शेषफल** को Return करता है। जैसे

```
C = A % B
```

```
C = 10 % 3
```

```
C = 1
```

हम विभिन्न प्रकार के Arithmetical Operators को निम्न Program द्वारा Use करके उनके काम करने के तरीके को समझ सकते हैं:

Program

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

main()
{
    int A = 10, B = 3, C;

    C = A + B ;
    printf("\n Addition    = %d", C);
    C = A - B ;
    printf("\n Subtraction  = %d", C);
    C = A * B ;
    printf("\n Multiplication = %d", C);
    C = A / B ;
    printf("\n Division      = %d", C);
    C = A % B ;
    printf("\n Modules|Reminder = %d", C);
    getch();
}
```

Output:

```
Addition    = 13
Subtraction  = 7
Multiplication = 30
Division      = 3
Modules|Reminder = 1
```

जब भी हम किसी प्रकार की कोई Calculation करते हैं, Calculation के बाद किसी ना किसी प्रकार का कोई मान Generate होता है। इस मान को Hold करने के लिए हम हमेशा किसी तीसरे Identifier को Use करते हैं।

जब हमें किसी Calculation से Generate होने वाले मान को किसी Identifier में Store करना होता है, तब हम उस Target Identifier को **Equal To (=)** Symbol के Left Side में लिखते हैं और Result Generate करने वाली Calculation में भाग ले रहे Identifiers के Expression को Equal To Symbol के Right Side में Specify करते हैं। Equal To Symbol को "C" Language में **Assignment Operator** कहा जाता है।

ये Operator अपने Right Side में Perform होने वाली Calculation से Generate होने वाले Resulting मान को अपने Left Side के Identifier में Store करने का काम करता है।

इस Program में सबसे पहले Variable Identifier **A** व **B** के बीच Addition, Subtraction आदि की प्रक्रिया होती है, जिससे कोई ना कोई Resultant मान Generate होता है। मान Generate होने के बाद उस मान को **Equal To Operator** Identifier **C** में Assign कर देता है, यानी Resultant मान Identifier **C** में Store हो जाता है। फिर printf() Function द्वारा Identifier **C** में Stored इस Resultant मान को Output में Display कर दिया जाता है।

Exercise:

- 1 Operator किसे कहते हैं ? Unary व Binary Operators के बीच क्या अन्तर है?
- 2 Initialization व Assignment के बीच के अन्तर को एक उदाहरण द्वारा समझाईए।

Relational Operators

Real World यानी वास्तविक जीवन में भी हम हमेंशा देखते हैं कि हर काम के साथ किसी ना किसी तरह की कोई शर्त जरूर **Associated** होती है। उदाहरण के लिए लोग आसानी से चल सकें, इसके लिए **Road** बनाया जाता है। लेकिन लोग रोड के बीच में नहीं चल सकते हैं। रोड पर चलने के साथ शर्त ये है कि लोगों को हमेंशा **Road** के **Left Side** में ही चलना चाहिए।

ठीक इसी तरह से जब हम कोई **Program Develop** करते हैं, तब हमेंशा ये जरूरी नहीं होता है कि विभिन्न प्रकार के काम करने के लिए सभी **Statements** को एक क्रम में ही **Execute** करना होगा। कई बार ऐसी परिस्थितियां होती हैं, जिनमें किसी एक परिस्थिति में किसी एक **Statement** को **Execute** करना होता है, जबकि दूसरी परिस्थिति में किसी अन्य **Statement** को **Execute** करने की जरूरत होती है।

यानी शर्त (**Condition**) के आधार पर एक ही **Program** में एक ही **Control** को एक **Statement** से दूसरे **Statement** पर भेजने की जरूरत पड सकती है। ठीक इसी तरह से किसी एक ही **Statement** को किसी विशेष परिस्थिति (**Condition**) में बार-बार **Execute** करना पड सकता है। **Programming** में इस प्रकार की **Situations** को **Handle** करने के लिए कुछ अन्य **Operators** को **Define** किया गया है, जिन्हें **Relational Operators** कहते हैं।

जब प्रोग्राम में किसी शर्त के आधार पर दो अलग **Statements** को **Execute** करने की जरूरत होती है, जहां पहली स्थिति में किसी एक **Statement** को **Execute** करना होता है, जबकि दूसरी स्थिति में किसी दूसरे **Statement** को **Execute** करना होता है, तब इस परिस्थिति में दो अलग मानों की आपस में तुलना की जाती है। तुलना करने पर यदि पहली **Condition** सही होती है, तो पहले **Statement** को **Execute** किया जाता है, जबकि पहली **Condition** गलत होने की स्थिति में किसी दूसरे **Statement** को **Execute** किया जाता है।

जब **Program** में किसी **Condition** के आधार पर **Execute** होने वाले **Statements** का चुनाव करना होता है, तब **Condition** को **Specify** करने के लिए हम इन **Relational Operators** का प्रयोग करते हैं। किसी प्रोग्राम में इन **Operators** का प्रयोग करके हम ये पता लगाते हैं कि कोई **Condition** सही है या नहीं। यदि **Statement** सही (**True**) होती है, तो ये **Operators 1** Return करते हैं और यदि **Condition** सही नहीं होती है (**False**) तो ये **Operators 0** Return करते हैं। **Relational Operators** निम्न हैं:

Operator	Mathematical Symbol	"C" Symbol
Equal to	=	==
Not Equal to	<>	!=
Less then	<	<
Greater then	>	>
Less then or Equal to	<=	<=
Greater then or Equal to	>=	>=

ये Relational Operators के काम करने के तरीके को हम निम्न Program द्वारा समझ सकते हैं। इस Program में हम देख सकते हैं कि जब Condition **True** होती है, तब **1** Return होता है और जब Condition **False** होती है, तब **0** Return होता है। Condition के आधार पर जिस प्रकार का मान Return होता है, उसे हमने इस Program में Output में Display किया है।

Program

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

main()
{
    printf("\n 10 is equal to 10 [10 == 10] : %d", 10==10);
    printf("\n 10 is less than 100 [10 < 100] : %d", 10<100);
    printf("\n 100 is greater than 10 [100 > 10] : %d", 100>10);
    printf("\n 10 is less than or equal to 10 [10 <= 10] : %d", 10<=10);
    printf("\n 10 is greater than or equal to 10 [10 >= 10] : %d", 10>=10);
    printf("\n 11 is not equal to 10 [11 != 10] : %d", 11!=10);

    putchar('\n');

    printf("\n 10 is equal to 11 [10 == 11] : %d", 10==11);
    printf("\n 100 is less than 10 [100 < 10] : %d", 100<10);
    printf("\n 10 is greater than 100 [10 > 100] : %d", 10>100);
    printf("\n 11 is less than or equal to 10 [11 <= 10] : %d", 11<=10);
    printf("\n 10 is greater than or equal to 11 [10 >= 11] : %d", 10>=11);
    printf("\n 10 is not equal to 10 [10 != 10] : %d", 10!=10);
}
```

Output:

```
10 is equal to 10 [10 == 10] : 1
10 is less than 100 [10 < 100] : 1
100 is greater than 10 [100 > 10] : 1
10 is less than or equal to 10 [10 <= 10] : 1
10 is greater than or equal to 10 [10 >= 10] : 1
11 is not equal to 10 [11 != 10] : 1

10 is equal to 11 [10 == 11] : 0
100 is less than 10 [100 < 10] : 0
10 is greater than 100 [10 > 100] : 0
11 is less than or equal to 10 [11 <= 10] : 0
10 is greater than or equal to 11 [10 >= 11] : 0
10 is not equal to 10 [10 != 10] : 0
```

हालांकि इस Program को सरल बनाए रखने के लिए हमने Literals का प्रयोग किया है। लेकिन यदि हम चाहें तो विभिन्न प्रकार के मानों को विभिन्न प्रकार के Variables या Constant Identifiers में Store करके उन Identifiers की भी आपस में तुलना कर सकते हैं। ऐसा करने पर भी प्राप्त होने वाले परिणाम में किसी प्रकार का कोई Change नहीं होता है।

उदाहरण के लिए यदि इसी Program में हम तीन Integer प्रकार के Variables **A**, **B**, व **C** Create करें और उनमें क्रमशः **10**, **11**, व **100** Store कर दें, और फिर पिछले Program में हमने

जहां-जहां Integer Literal **10** को Use किया है, वहां Identifier **A** को, जहां-जहां Integer Literal **11** को Use किया है, वहां-वहां Identifier **B** को व जहां-जहां Integer Literal **100** को Use किया है, वहां-वहां Identifier **C** को Replace कर दें, तो भी हमें प्राप्त होने वाला Output वही प्राप्त होगा, जो इस Program से प्राप्त हो रहा है।

Ex/ercise:

- 1 Relations Operators को समझाईए। ये Operators किस तरह से काम करते हैं और इनका प्रयोग क्यों किया जाता है?
- 2 एक Program में विभिन्न प्रकार के Relational Operators को Use कीजिए व समझाईए कि ये किस प्रकार से किसी Condition के आधार पर True या False Return करते हैं।

Conditional Operators / Ternary Operator

यह **if . . . else** Conditional Statement का संक्षिप्त रूप है, जिसके बारे में हम अगले Chapter में विस्तार से पढ़ेंगे। इसका Syntax निम्नानुसार होता है:

$$\text{Target} = (\text{Condition}) ? A : B$$

इस Operator को Use करने पर यदि Braces में दी गई Condition से True Return होता है, तो Target Identifier में Identifier **A** का मान Store हो जाता है। लेकिन यदि Braces में दी गई Condition **True** के स्थान पर **False** Return करता है, तो Target Identifier में Identifier **B** का मान Store हो जाता है।

यदि हम दो संख्याओं में से बड़ी संख्या प्राप्त करने का Algorithm बनाना चाहें, तो ये Algorithm हम निम्नानुसार बना सकते हैं, जहां Identifier **A** व Identifier **B** वे मान हैं, जिनकी आपस में तुलना करनी है और Target Identifier वह Identifier है, जो Condition के आधार पर Identifier A या Identifier B से Return होने वाले मान को Hold करता है।

Algorithm:

CONDITIONAL_OPERATOR(A, B, TARGET)

IF A is greater than B THEN

TARGET = A

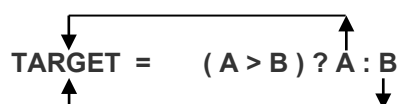
ELSE

TARGET = B

इसी Algorithm के आधार पर यदि हम Ternary Operator को Define करें, तो हमें Ternary Operator के लिए निम्नानुसार Algorithm प्राप्त होता है:

$$\text{TARGET} = (A > B) ? A : B$$

मानलो यदि $A = 2$, $B = 3$ व $\text{TARGET} = ?$ हो, तो Ternary Operator में इन Identifiers को Place करने पर हमें TARGET Identifier में **3** प्राप्त होगा, क्योंकि Condition $(A > B)$ के Execute होने पर **True** यानी **1** Return होगा और Condition के **False** होने की स्थिति में Identifier **A** का मान Target में Store हो जाएगा। यदि एक चित्र द्वारा Ternary Operator के काम करने के तरीके को Represent करें, तो बनने वाला चित्र निम्नानुसार होगा:



यदि Ternary Operator का प्रयोग करके हम एक ऐसा Program बनाना चाहें, जो दो संख्याओं में से बड़ी संख्या को प्राप्त करके Output में Print करें, तो हम ये Program निम्नानुसार बना सकते हैं:

Program

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

main()
{
    int num1 = 10, num2 = 30, big;

    big = num1 > num2 ? num1 : num2;

    printf("Biggest Number in %d and %d is = %d", num1, num2, big);

    getch();
}
```

Output:

Biggest Number in 10 and 30 is = 30

इस Program में हमने printf() Function को अलग तरीके से Use किया है और तरीके का Effect हम Program के Output में देख सकते हैं। Function के पहले Control String के स्थान पर **num1** का, दूसरे Control String के स्थान पर **num2** का व तीसरे Control String के स्थान पर Variable Identifier **big** का मान Print हो रहा है।

सामान्यतया जब हम किसी Variable Type के Identifier को Declare करते हैं, तब उसे Variable नाम से ही सम्बोधित करते हैं और जब हम **const** Keyword का प्रयोग करके Constant Identifier Declare करते हैं, तब उसे केवल Constant नाम से ही सम्बोधित करते हैं। इसलिए अब यहां से हम भी इस प्रकार के Identifiers को इन्हीं सम्बोधनों को उपयोग में लेंगे।

इस Program में printf() Function को इस तरह से Use करने का कारण ये है, कि यदि हम इसी Program में num1 का मान बदल कर **20** व num2 का मान बदल कर **10** कर दें, यानी Program के main() Function के सबसे पहले Statement को यदि हम निम्नानुसार Modify कर दें:

```
int num1 = 20, num2 = 10, big;
```

तो हमें इसी Program से प्राप्त होने वाला Output निम्नानुसार प्राप्त होगा, जो कि पिछले Program के Output से अलग है व ज्यादा सभी तरीके से Information दे रहा है:

Output:

Biggest Number in 20 and 10 is = 30

यानी printf() Function को इस तरह से Use करके हम एक ही Printf() Function द्वारा अलग-अलग प्रकार के Output प्राप्त कर सकते हैं।

Software Programming के क्षेत्र में हमेशा दो तरह के लोग होते हैं। एक वे जो विभिन्न प्रकार की समस्याओं को Solve करने के लिए विभिन्न प्रकार के Programs Develop करते हैं। इस प्रकार के लोगों को **Software Programmer** कहा जाता है, जबकि दूसरे प्रकार के लोग वे लोग होते हैं, जो किसी समस्या का समाधान प्राप्त करने के लिए पहले प्रकार के लोगों यानी Programmers द्वारा Develop किए गए Programs को Use करते हैं। Programs को Use करने वाले इस प्रकार के लोगों को **Program User** कहा जाता है।

एक Software Programmer कभी भी किसी User को अपने Source Codes नहीं देता है, बल्कि वह User को केवल Executable Codes प्रदान करता है, ताकि User उसके Program को Use तो कर सके, लेकिन उसमें किसी प्रकार का कोई Modification ना कर सके।

यदि हम इस तरह से सोचें कि हम एक Programmer हैं और हम जो Program बना रहे हैं, उसे कोई User केवल Use ही कर सकेगा, तो हमने अभी तक जितने भी Programs बनाए हैं, उन में से किसी भी Program से User को कोई फायदा नहीं होने वाला है। क्योंकि अभी तक हमने कोई भी ऐसा Program नहीं बनाया है, जिसमें User अपनी जरूरत के आधार पर किसी प्रकार का कोई Result प्राप्त कर सके।

उदाहरण के लिए यदि हम हमारे पिछले Program को ही लें, तो इस Program से उसी स्थिति में दो अलग संख्याओं का Comparison करके सबसे बड़ी संख्या को Output में Display किया जा सकता है, जब दोनों मानों को Program के Source Codes में Modify करके Program को फिर से Recompile किया जाए, जबकि Source Codes तो हम किसी User को देंगे ही नहीं।

इसलिए ये Program किसी User के लिए तब तक बेकार है, जब तक कि Program के Run होते समय User की जरूरत के आधार पर ये Program User से Run Time में Data प्राप्त करके उन्हें Process ना कर सके और सबसे बड़ी संख्या को Output में Display ना कर सके।

यदि हम सारांश में कहें तो अभी तक के जितने भी Programs हमने बनाए हैं, उन में से किसी भी Program में Input की सुविधा को हमने Add नहीं किया है और बिना Input की सुविधा के एक Program केवल उन्हीं मानों के साथ प्रक्रिया कर सकता है, जिन्हें Program Develop करते समय Initialize किया गया होता है। ऐसे Programs हमेशा एक ही प्रकार का Output प्रदान करते हैं। ये Programs Game की तरह होते हैं, जो हमेशा एक ही तरह से Run होते हैं।

Program में Interactivity लाने के लिए व एक ही Program द्वारा विभिन्न प्रकार के मानों के साथ प्रक्रिया करने की क्षमता प्राप्त करने के लिए Program में Input की सुविधा को भी Add करना जरूरी होता है। "C" Language में जिस तरह से Output को Display करने के लिए printf() Function को **stdio.h** नाम की Header File में Define किया गया है, ठीक उसी तरह से Input की सुविधा को प्राप्त करने के लिए **scanf()** नाम का एक Function भी इसी Header File में Define किया गया है।

scanf() Function

Computer से जितने भी Devices Connected होते हैं, उन सभी Devices की अपनी स्वयं की Memory होती है, जिसे **Temporary Buffer** कहा जाता है। Keyboard, Monitor, Mouse, Printer आदि सबका अपना Temporary Buffer होता है। हम Keyboard से जब भी किसी Key को Press करते हैं, तो उस Key की Information Directly Computer की RAM में जा कर Store नहीं होती है, बल्कि उस Key की Information सबसे पहले Keyboard के Memory Buffer में Store होती है, जहां से हमारे Computer का CPU उस Key की Information को Computer के RAM में Store करता है।

ठीक इसी तरह से जब हम हमारे Computer के Monitor पर किसी Message को Print करना चाहते हैं, तो वास्तव में हम Message को Print करने के लिए Computer के Monitor नहीं भेज रहे होते हैं, बल्कि हम उस Printable Message को Computer के **Graphics Buffer** में भेज रहे होते हैं, जहां से हमारा Monitor Printable Message की Information को प्राप्त करके Monitor पर Display कर देता है।

जब हम `printf()` Function को Use करके किसी Message को Monitor पर Display करना चाहते हैं, तब हम उस Message या Data को `printf()` Function में एक String Argument के रूप में भेज देते हैं। `printf()` Function उस String Message को Computer की Memory से प्राप्त करके Monitor के **Graphics Buffer** में Store देता है और इस Graphics Buffer में Stored Data को हमारा Monitor अपने Screen पर Display कर देता है।

इसी तरह से जब हम Keyboard से किसी Input को प्राप्त करना चाहते हैं, **`scanf()`** Function Keyboard पर Press की गई Keys की Information को Keyboard के **Buffer** से प्राप्त करता है और उन Keys की Information को **`scanf()`** Function में Specify किए गए Variable Identifier की Storage Location पर Store कर देता है।

दूसरे शब्दों में कहें तो जब हम Memory में Stored किसी Data को Monitor पर Display करना चाहते हैं, तब `printf()` Function में विभिन्न Identifiers को Specify करके हम हमारे Computer को ये बताते हैं कि हमें Memory की किस Location पर Stored Data को Screen पर Display करना है और विभिन्न प्रकार के Control Strings का प्रयोग करके हम हमारे Computer को ये बताते हैं कि विभिन्न Identifiers द्वारा Specify किए जा रहे Data को Monitor के Screen की किस Location पर व किस Format में Display करना है।

इसी तरह से जब हम Keyboard से किसी Data को Input के रूप में प्राप्त करके किसी Memory Location पर Store करना चाहते हैं, तब जिस Data Type के Data को Keyboard से Receive करना चाहते हैं, उस Data Type के Control String को **`scanf()`** Function में Specify करते हैं और Keyboard से आने वाले Data को Memory के जिस Storage Location पर Store करना चाहते हैं, **`scanf()`** function में उस Storage Location के Variable Identifier का नाम Address Operator (**&**) के साथ Specify करते हैं।

जिस तरह `printf()` Function से साथ हम विभिन्न प्रकार के Control Strings का प्रयोग करके विभिन्न प्रकार के Identifiers के मानों को Output में Print करते हैं, उसी तरह से विभिन्न प्रकार के Data Type के मानों को Keyboard Buffer से प्राप्त करके विभिन्न प्रकार के Identifiers में Store करने के लिए भी हम विभिन्न प्रकार के Control Strings का प्रयोग कर सकते हैं। `printf()` Function के साथ जो Control String जिस Data Type से Related होता है, `scanf()` Function में भी वह Control String उसी Data Type से Associated होता है। `scanf()` Function के साथ Use किए जा सकने वाले Control Strings निम्नानुसार हैं:

- %d** Keyboard से Integer Data Type के मान को प्राप्त करने के लिए
- %c** Keyboard से Character Data Type के मान को प्राप्त करने के लिए
- %f** Keyboard से Floating Point Real Data Type के मान को प्राप्त करने के लिए
- %g** Keyboard से Floating Point Real Data Type के मान को प्राप्त करने के लिए
- %e** Keyboard से Floating Point Real Data Type के मान को प्राप्त करने के लिए
- %i** Keyboard से Signed Decimal Integer Data Type के मान को प्राप्त करने के लिए
- %u** Keyboard से Unsigned Decimal Integer Data Type के मान को प्राप्त करने के लिए
- %o** Keyboard से Octal Integer Data Type के मान को प्राप्त करने के लिए

%s Keyboard से String को प्राप्त करने के लिए
%x Keyboard से Hexadecimal Data Type के मान को प्राप्त करने के लिए
%[...] Keyboard से String को प्राप्त करने के लिए

scanf() Function **printf()** function की तुलना में एक अधिक Control String को Support करता है। **scanf()** function का Syntax निम्नानुसार होता है:

Syntax:

```
scanf("cntrlStr1 cntrlStr2 cntrlStrN", &Identifier1, &Identifier2, &IdentifierN)
```

इस Syntax में **cntrlStr** ये तय करती हैं कि Keyboard से किसी Data Type का Data **scanf()** function Receive करेगा, जबकि **Identifier** उस Memory Location को Represent करता है, जहां पर Keyboard से आने वाले Data को Store करना है। इस Function में भी Control Strings जिस क्रम में Specify किए जाते हैं, उसी क्रम में आने वाले Data भी Memory में Store होते हैं।

उदाहरण के लिए cntrlStr1 Identifier1 से, cntrlStr2 Identifier2 से व cntrlStrN IdentifierN से Associated है, इसलिए Keyboard से आने वाला cntrlStr1 Type का सबसे पहला मान Identifier1 की Storage Location पर Store होगा, दूसरे Number पर आने वाला cntrlStr2 Type का मान Identifier2 के Memory Location पर Store होगा और सबसे बाद में आने वाला cntrlStrN Type का मान IdentifierN नाम के Identifier द्वारा Represent होने वाली Memory Location पर Store होगा।

& Operator को **Address Operator** कहा जाता है। ये एक Unary Operator है। ये Operator हमेशा उस Identifier के Memory Location का Address Return करता है, जिसके साथ इसे Use किया जाता है।

जब हम Program के **Run Time** के Keyboard से किसी Data को Receive करके उस पर Processing करना चाहते हैं, तब **scanf()** Function द्वारा Computer को हमें दो बातें बतानी पड़ती हैं: पहली ये कि हम Keyboard से किस प्रकार के Data को Read करना चाहते हैं। Keyboard से Read किए जाने वाले Data के Data Type को Specify करने के लिए उपयुक्त Control String का प्रयोग किया जाता है।

Computer को दूसरी बात ये बतानी होती है, कि Keyboard से आने वाला Data Memory की किस Location पर Store होगा। यानी दूसरी बात के रूप में हमें Computer को उस Memory Location का Address बताना होता है, जहां पर हम Keyboard से आने वाले Data को Store करना चाहते हैं।

जैसाकि हमने पहले भी कहा कि **&** Operator किसी भी Identifier का Address Return करने का काम करता है, इसलिए हमें जिस Identifier की Memory Location पर Keyboard से आने वाले Data को Store करना होता है, उस Identifier के नाम के पहले हम **&** Operator का प्रयोग उस Identifier को **scanf()** Function में Specify कर देते हैं।

चूंकि **scanf()** Function का प्रयोग Keyboard से Input प्राप्त करने के लिए किया जाता है, इसलिए इस Function का प्रयोग करने से पहले हमें उस Data Type का एक Variable Identifier Create करना जरूरी होता है, जिसमें हमें **scanf()** Function द्वारा Keyboard से आने वाले Data को Store करना चाहते हैं।

बिना Variable Create किए हुए, हम scanf() Function का प्रयोग नहीं कर सकते हैं, क्योंकि इस Function में हमें उस Identifier का नाम Address Operator के साथ Specify करना पड़ता है, जिसकी Memory location पर हम Keyboard से आने वाले मान को Store करना चाहते हैं। यदि हम बिना Variable Create किए हुए scanf() Function को Use करते हैं, तो "C" का Compiler Compile Time Error Generate करके हमें ऐसा करने से रोक देता है।

चलिए, एक उदाहरण द्वारा scanf() Function को Use करना सीखते हैं। मानलो कि हम Keyboard से किसी Student की **Age** को Read करना चाहते हैं और उस Age में 10 जोड़कर Resultant मान को Screen पर Display करना चाहते हैं। इस समस्या का Algorithm निम्नानुसार बनाया जा सकता है:

Algorithm :

SIMPLE_INPUT(AGE, RESULT)

Where:

AGE is the age of student and

RESULT is the modified age of the student.

- | | | |
|---|---------------------------|--------------------------|
| 1 | START | [Start the program.] |
| 2 | READ AGE | [Get AGE from keyboard.] |
| 3 | PROCESS RESULT = AGE + 10 | |
| 4 | PRINT RESULT | |
| 5 | END | [End the program.] |

इसी Algorithm के आधार पर हम "C" भाषा में Program भी बना सकते हैं, जिसमें सबसे पहले हमें ये तय करना है कि हमें किस प्रकार का Data Computer की Memory में Process करने के लिए Store करना है। चूंकि Age एक **Unsigned Type** का मान होता है, जो कि कभी भी Minus में या Negative Type में नहीं हो सकता है, साथ ही Age एक ऐसा मान होता है, जो बहुत ही छोटा होता है, क्योंकि किसी भी व्यक्ति की Normal Age 100-150 साल से ज्यादा नहीं हो सकती है, इसलिए हम Age को Store करने के लिए Unsigned Character Type का Variable Identifier Create कर सकते हैं, क्योंकि इस Data Type के Identifier की Range 0 से 255 तक होती है, जिसमें किसी की भी Age आसानी से Store हो सकती है। अब यदि हम इस समस्या का "C" Program बनाना चाहें, तो ये Program निम्नानुसार होगा:

Program

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

main()
{
    /* Declaration Section */
    unsigned char age, result;
    clrscr();

    /* Input Section */
    printf("Enter Age of the student : ");
    scanf("%u", &age);

    /* Process Section */
    result = age + 10;
```

```

/* Output Section */
printf("After 10 year, student will be %u years old", result);

getch();
}

```

हमें एक "C" Program में जितने भी Identifiers को Use करना होता है, उन सभी Identifiers को Declaration Section में ही Declare करना जरूरी होता है। चूंकि हम हमारे इस Program में Keyboard से Input लेना चाहते हैं, इसलिए User को एक Message देकर ये बताना जरूरी होता है, कि Program को काम करने के लिए किस प्रकार के मान की जरूरत है।

इसीलिए Input Section में scanf() Function को Use करने से पहले हमने एक printf() Statement को Use करके User को Student की Age Input करने का Message दिया है। जब हम इस Program को Compile करके Run करते हैं, तो Program के Run होते ही User को निम्नानुसार ये Message दिखाई देता है और Data प्राप्त करने के लिए Cursor Blink करने लगता है:

```
Enter Age of the student : _
```

यदि printf() Statement द्वारा ये Message Print ना करें, तो Output में Black Screen पर केवल Cursor Blink करता हुआ ही दिखाई देता है और User को पता ही नहीं लगता कि उसे करना क्या है। जहां पर Cursor Blink कर रहा है, वहां पर User जो भी मान Input करता है, उस मान को scanf() Function Scan करता है।

मानलो User ने इस स्थान पर 15 Input किया, तो scanf() function इस 15 को Scan करेगा और इस मान को उस **age** नाम के Variable के Memory Location पर भेज देगा, जिसका नाम **&** Address Operator के साथ scanf() Function में Specify किया गया है।

scanf() Function जैसे ही Keyboard से आने वाले मान को Computer की उस Memory Location पर Store करता है, जिसका नाम **age** है, वैसे ही Input का काम समाप्त हो जाता है। उसके बाद Computer Program के अगले Statement को Execute करके Age में 10 जोड़ता है और इसके बाद के printf() Statement द्वारा Resultant मान को Screen पर निम्नानुसार Form में Print कर दिया जाता है:

```
After 10 year, student will be 25 years old
```

जब ये Program पूरी तरह से Run हो जाता है, तब इसका Output हमें निम्नानुसार प्राप्त होता है:

Output:

```
Enter Age of the student : 15
After 10 year, student will be 25 years old
```

अब यदि हम दो संख्याओं में से बड़ी संख्या निकालने वाले Program को Modify करना चाहें, जिसमें User स्वयं अपनी इच्छानुसार दोनों संख्याओं को Input करे और Program, Input की गई दोनों संख्याओं को Compare करके बड़ी संख्या को Output में Print करे, तो इस Program को Develop करने के लिए हम निम्न Algorithm का प्रयोग कर सकते हैं:

Algorithm:

BIG_IN_2(A, B, BIG) Where:
A is the first number.
B is the second number. and
BIG is the biggest number between A and B.

```
START
READ A, B          [Get values from keyboard to be compare.]
IF A is greater than B THEN [Process: Compare to get biggest. ]
    BIG = A
ELSE
    BIG = B
PRINT BIG          [Display biggest value on the monitor. ]
EXIT
```

इस Algorithm के आधार पर यदि हम "C" Program बनाना चाहें, तो Program को निम्नानुसार बनाया जा सकता है:

Program

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

main()
{
    /* Declaration Section */
    long double A, B, BIG;

    /* Input Section */
    printf("Enter first value :");
    scanf("%Lf", &A);
    printf("Enter second value :");
    scanf("%Lf", &B);

    /* Process Section */
    BIG = (A > B) ? A : B;

    /* Output Section */
    printf("\n Biggest Value is : %Lf", BIG);
    getch();
}
```

Output:

```
Enter first value : 12457889562312.323232
Enter second value : 1223564574898.121212

Biggest Value is : 12457889562312.323230
```

इस Program में हमने **long double** Type के Variable Declare किए हैं, इसलिए इनमें Value Input करने के लिए हमें **scanf()** Function **%Lf** Control String की जरूरत होती है। इसी तरह

से इन Variables में Stored Values को Display करने के लिए भी हमें printf() Function में %Lf Control String को Use करना होता है।

यदि हम इन Identifiers में घातांक रूप में मानों को Input करना चाहें या फिर इन Identifiers में Stored मानों को Output में Display करने के लिए घातांक रूप का प्रयोग करना चाहें तो, दोनों ही स्थितियों में हमें %Lf Control String के स्थान पर %Le Control String का प्रयोग करना जरूरी होता है।

इस Program में हमने long double प्रकार के Identifiers इसलिए लिए हैं, ताकि हम बड़ी से बड़ी संख्या को इसमें Store कर सकें। इस Program में भी Program के Run Time में Keyboard से Input प्राप्त करने के लिए हमने उसी Process को Use किया है, जिस Process को पिछले Program में Use किया था।

यानी सबसे पहले एक printf() Statement द्वारा User को ये Message प्रदान किया है, कि वह पहला मान Input करे। फिर scanf() Function का प्रयोग करके User द्वारा प्रदान किए गए Input को Accept करके उस A नाम के Variable में Store किया जिसका प्रयोग & Address Operator के साथ किया गया है। इसी तरह से एक और Message दे कर दूसरे Variable के लिए भी User से Input प्राप्त किया।

scanf() Function का प्रयोग करके हम एक ही बार में एक से ज्यादा Variables में मान Store कर सकते हैं। scanf() Function को इस प्रकार से Use करने की जरूरत तब पड़ती है, जब कई मान एक साथ एक Group के रूप में किसी विशेष सूचना को Represent करते हैं।

उदाहरण के लिए यदि Keyboard से Date या Time Input करना हो, तो Date या Time को हम अलग-अलग टुकड़ों में Input नहीं कर सकते हैं। ऐसे Group of Data को हमें एक साथ Input करना होता है। एक Date या Time में हमेंशा तीन हिस्से होते हैं, जो क्रमशः Day, Month, Year या Hour, Minutes, Seconds को Represent करते हैं। Keyboard से जब इस प्रकार के Data को Read करना होता है, तब Data या Time Input करने का एक ही Message दिया जाता है और तीनों मानों को एक साथ Input कर दिया जाता है।

निम्न Program द्वारा scanf() Function को इस प्रकार Use करने की कार्य-विधि को ज्यादा अच्छे तरीके से समझा जा सकता है। ये Program User से उसकी Date Of Birth (DOB) व Current Date Input करने के लिए कहता है। जब User उसकी DOB व Current Date Input कर देता है, तब Program Output के रूप में उस User की Current Age Display करता है।

Program

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

main()
{
    /* Declaration Section */
    int dob_DD, dob_MM, dob_YYYY;
    int cur_DD, cur_MM, cur_YYYY;
    char dummy;
    int age;

    /* Input Section */
    printf("Enter Date of Birth in DD/MM/YYYY Format : ");
```

```
fflush(stdin);
scanf("%2d%1c%2d%1c%4d", &dob_DD, &dummy, &dob_MM, &dummy, &dob_YYYY);

printf("Enter Today's Date in DD/MM/YYYY Format : ");
fflush(stdin);
scanf("%2d%1c%2d%1c%4d", &cur_DD, &dummy, &cur_MM, &dummy, &cur_YYYY);

/* Process Section */
age = cur_YYYY - dob_YYYY;

/* Output Section */
printf("\n Your Date of Birth is : ");
printf("%d%c%d%c%d", dob_DD, '/', dob_MM, '/', dob_YYYY);
printf("\n And you are %d years old now", age);
}
```

Output:

```
Enter Date of Birth in DD/MM/YYYY Format : 06/03/1982
Enter Today's Date in DD/MM/YYYY Format : 11-04-2008

Your Date of Birth is : 6/3/1982
And you are 26 years old now
```

Description:

इस Program को Run करते ही ये Program हमें DD/MM/YYYY Format में Birth Date Input करने के लिए कहता है। जैसे ही हम Birth Date Input करते हैं, ये Program DD/MM/YYYY Format में ही हमसे Current Date Input करने के लिए कहता है। जैसे ही हम Current Date भी Input करते हैं, ये Program हमें हमारा **Date Of Birth** व हमारी Current Age Screen पर Display कर देता है।

इस Program में हमने कई नए Concepts Use किए हैं, लेकिन ये Program पूरी तरह से **Error Proof** नहीं है, क्योंकि ये जिस Format में Date Input करने के लिए कहता है, हमें उस Format को पूरी तरह से Follow करना पड़ता है। यानी हम Date Of Birth **06/03/1982** को **6/3/1982** Format में Input नहीं कर सकते हैं। यदि हम ऐसा करते हैं, तो हमारा Program हमें सही Output नहीं देता है।

जैसाकि हम देख सकते हैं, कि **Date** एक ऐसा **Data** है, जिसके होते तो तीन हिस्से हैं, लेकिन इसके तीनों हिस्सों को एक ही बार में Input करना जरूरी होता है। हम देख सकते हैं कि इस **Date** में **Day**, **Month** व **Year** के अलावा एक और चौथा हिस्सा भी है, जो **Day**, **Month** व **Year** को आपस में एक दूसरे से अलग रखता है।

scanf() Function जब एक ही बार में एक से ज्यादा मानों को Input के रूप में प्राप्त करना चाहता है, तब एक **scanf()** द्वारा जितने **Data** Computer की **Memory** में **Store** करने होते हैं, उन सभी मानों के **Control Strings** के साथ उनके **Variables** को **scanf()** Function में ठीक उसी तरह से **Specify** किया जाता है, जिस तरह से **printf()** Function द्वारा एक से अधिक **Identifiers** के मानों को **Output** में **Display** करने के लिए किया जाता है।

इन दोनों Functions में अन्तर केवल इतना होता है कि **scanf()** Function में **Specify** किए जाने वाले सभी **Identifiers** **Keyboard Buffer** से अपना मान प्राप्त करते हैं, और सभी मानों को

Variables की Reserved Memory Location पर भेजने के लिए इन Variables के साथ Address Operator का प्रयोग किया जाता है।

इस Program में हमने scanf() Function में Use किए जाने वाले Control Strings को थोड़ा अलग तरीके से Use किया है। scanf() Function के इस तरीके से Input लेने की प्रक्रिया को **Formatted Input** कहते हैं।

चूंकि एक Date के पहले दो अंक Day को Represent करते हैं, इसलिए Input किए जाने वाले Date के पहले दो Characters को ही हमें **dob_DD** व **cur_DD** Variable में Store करना होता है।

इस जरूरत को पूरा करने के लिए हमने पहले Control String के साथ एक Digit 2 का प्रयोग %2s के रूप में किया है। जब हम इस तरह से Control String Use करते हैं, तब Compiler Keyboard से आने वाले Input में से केवल पहले दो अंकों को ही dob_DD व cur_DD में Store करता है।

चूंकि तीसरा Character एक Separator के रूप में काम कर रहा है जो Day को Month की Digit से अलग करता है, इसलिए %1c Control String का प्रयोग करके इस तीसरे Character को हमने dummy नाम के एक Character प्रकार के Variable में Store कर दिया है।

अब Input के रूप में आने वाले अगले दो Digits Month को Represent करते हैं। केवल इन दो Digits को प्राप्त करके **dob_MM** व **cur_MM** में Store करने के लिए हमने फिर से %2d का प्रयोग किया है और Month को Year से Separate करने वाले Separator को फिर से %1c Control String द्वारा dummy नाम के Variable में Store कर लिया है। फिर अन्तिम 4 Digits को **dob_YYYY** व **cur_YYYY** Variable में Store करने के लिए हमने %4d Control String का प्रयोग किया है।

इस Program में हमने निम्नानुसार एक Statement का scanf() Function से पहले प्रयोग किया है:

```
fflush(stdin);
```

ये Function एक विशेष काम करता है। जब हम Keyboard से Keys को Press करते हैं, तब जरूरत के आधार पर विभिन्न Characters विभिन्न Variables में Store हो जाते हैं। लेकिन कई बार जब हम Formatted Input का प्रयोग करते हैं, तब Keyboard से चाहे जितने Characters Input किए जाएं, Variable में Control String में Use किए गए मान के अनुसार कुछ ही Characters Store होते हैं, शेष Characters Keyboard के Buffer में ही पड़े रहते हैं।

यदि हम Keyboard के Buffer में पिछले Input के बचे हुए Characters को Clear किए बिना ही scanf() Function को Use करते हैं, तो कई बार scanf() Function User से कोई मान Input करने के लिए नहीं कहता है, बल्कि Keyboard के Buffer में Stored Characters को ही Use कर लेता है, जिससे Program का Output सही नहीं आता। इस स्थिति में ये Statement Keyboard के Buffer में Stored बचे हुए Characters को Clear करने का काम करता है, ताकि User को सही Output प्राप्त हो।

कई बार हमें ऐसी जरूरत भी पड़ जाती है, जिसमें हम एक ही scanf() Function द्वारा एक से ज्यादा Variables में मान तो Store करना चाहते हैं, लेकिन किसी Formatted Input Process को Use करना नहीं चाहते हैं। इस स्थिति में विभिन्न Identifiers में मानों को Store करने के लिए भी

scanf() Function को तो समान तरीके से ही Use किया जाता है। अन्तर केवल इतना होता है कि scanf() Function में Use किए जाने वाले Control Strings को Simple ही रखा जाता है।

चूँकि scanf() Function की एक विशेषता ये है कि ये Function Blank Space से Terminate हो जाता है। इसलिए यदि हम किसी मान को Input करते समय Space या Enter Key द्वारा कई मानों को अलग-अलग कर दें, तो Input किया गया मान scanf() Function में Specify किए गए विभिन्न Variables में Store हो जाते हैं। उदाहरण के लिए हम यहां दो संख्याओं को जोड़ने का एक Program बना रहे हैं, जिसमें एक ही scanf() Function द्वारा दोनों मानों को Input किया जा रहा है।

Program

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

main()
{
    /* Declaration Section */
    int firstVal, secondVal, result;

    /* Input Section */
    printf("Enter First and Second Values ");
    scanf("%d%d", &firstVal, &secondVal);

    /* Process Section */
    result = firstVal + secondVal;

    /* Output Section */
    printf("\n Total of %d and %d is = %d ", firstVal, secondVal, result);

    getch();
}
```

Output 1st :

```
Enter First and Second Values : 10 20 (Blank Space between values)
Total of 10 and 20 is 30
```

Output 2nd :

```
Enter First and Second Values : 10 (Pressed Enter between values)
20
Total of 10 and 20 is 30
```

Exercise:

- 1 printf() व scanf() Function के अन्तर व समानताओं का वर्णन कीजिए, तथा दोनों की कार्यप्रणाली को एक उचित उदाहरण द्वारा विस्तार से समझाईए।
- 2 एक Program बनाईए जो Input के रूप में User से Year प्राप्त करे और Output के रूप में User को ये बताए कि Input किया गया Year Leap Year है या नहीं। Leap Year एक ऐसा Year होता है, जिसमें हर चौथे साल February 29 दिन की होती है।

- 3 एक Program बनाईए जिसमें Input के रूप में User से एक अंक प्राप्त करता है और Output के रूप में User को ये बताता है कि Input किया गया मान सम है या विषम। जिस संख्या में दो का भाग पूरा-पूरा चला जाता है, वह संख्या सम संख्या होती है।
- 4 किसी शहर के Temperature को Centigrade के रूप में Input करो और इस मान के Fahrenheit मान को Output में Print करो।

$$(Fahrenheit = 1.8 * Centigrade + 32;)$$

- 5 एक Program बनाईए जिसमें User **Days** की संख्या Input करता है और Program उन Days को **Month** व **Reminder Days** में Convert करता है। उदाहरण के लिए यदि User Program में 50 Input करता है, तो Output में “1 Month 20 Days” Display होना चाहिए।
- 6 दो राज्यों के बीच की दूरी को KM में Input करो और इस Input किए गए इस मान को Meters, Centimeters, Feets व Inches के रूप में Convert करके Screen पर Print करो।
- 7 किसी Employee की Basic Salary Keyboard से Input करो और इस Basic Salary के आधार पर 20% Provident Fund(PF), 30% Dearness Allowance (DA) व 15% House Rent Allowance (HRA) Calculate करो। अब इस Calculation से प्राप्त Result के आधार पर उस Employee की Gross Salary ज्ञात करो। जब Basic Salary में विभिन्न प्रकार के Allowances, Funds आदि को जोड़ दिया जाता है, तब प्राप्त होने वाली Salary को **Gross Salary** कहते हैं।
- 8 एक आयत की लम्बाई व चौड़ाई Keyboard से Input करो और इस आयत का क्षेत्रफल तथा परिमाप ज्ञात करने का Program Algorithm की मदद से बनाओ।
- 9 एक वृत्त का क्षेत्रफल व परिमाप ज्ञात करने का Algorithm बनाओ और इस Algorithm के आधार पर Program Create करो जबकि Circle का Radius Keyboard से Input किया जाए।
- 10 किसी त्रिभुज का क्षेत्रफल ज्ञात करने का Program बनाओ जिसकी भुजाएं क्रमशः A, B व C हैं तथा त्रिभुज का क्षेत्रफल ज्ञात करने का सूत्र निम्नानुसार है:

$$\text{Area} = \sqrt{S(S-A)(S-B)(S-C)}$$

$$\text{Where } S = (A + B + C) / 2$$

- 11 Keyboard से एक चार Digit की संख्या Input करो और उस संख्या के First व Last Digit के योग को Output में Print करने का Program लिखें
- 12 यदि Keyboard से दो संख्याओं को Input किया जाए, तो दोनों संख्याओं को Exchange या Swap करने का Program लिखो। साथ ही इस Program के Algorithm को विस्तार से समझाओ। जब दो मानों को आपस में एक दूसरे के स्थान पर Exchange करके Store किया जाता है, तो इस प्रक्रिया को **Swapping** करना कहते हैं।

Logical Operators

Program में कोई ऐसी स्थिति होती है, जिसमें किसी एक Condition के आधार पर दो में से किसी एक काम को पूरा करना होता है, तब हम Relational Operators का प्रयोग करके Conditions को Check करते हैं। ,

लेकिन कई बार Program में ऐसी परिस्थितियां बन जाती हैं, जिसमें एक से अधिक Conditions के आधार पर किसी एक काम को पूरा करना होता है। जब किसी Program में इस प्रकार की परिस्थिति पैदा हो जाती है, जिसमें दो या दो से अधिक Conditions के साथ प्रक्रिया करके परिणाम प्राप्त करना होता है, तब Logical Operators का उपयोग किया जाता है।

“C” Language में मुख्यतः तीन Logical Operators होते हैं। चूंकि Logical Operators Binary Operators होते हैं, इसलिए इन Operators के साथ हमेशा दो Operands होते हैं, साथ ही Logical Operators जिन दो Operands के आधार पर Operation Perform करके Result Generate करना चाहते हैं, उनमें भी कोई ना कोई Relational Operator Included होता है।

AND (&&)

जब Logical Operator के दोनों तरफ की Condition **True** होती है, तब ये Logical Operator **True** या **1** Return करता है। यदि Logical Operator के दोनों तरफ की Conditions में से किसी एक भी Condition द्वारा **0** या **False** Return हो रहा हो, तो ये Logical Operator भी **False** Return करता है। जैसे:

```
X = (10 > 5) && (5 > 3)
```

ये Statement Identifier X में 1 यानी True Store करेगा, क्योंकि इस Statement के Execute होने पर सबसे पहले Logical AND Operator के Left Hand Side में स्थित Expression (10 > 5) Execute होगा, जो केवल उस स्थिति में True Return करता है, जब 10 का मान 5 के मान से बड़ा होता है।

चूंकि 10 हमेशा ही 5 से बड़ा होता है, इसलिए ये Expression True Return करता है। फिर Logical AND Operator के Right Side की Condition (5 > 3) Check होती है, जो उस स्थिति में True या 1 Return करता है, जब 5 का मान 3 से ज्यादा होता है।

चूंकि यहां भी 5 का मान हमेशा ही 3 से ज्यादा होता है, इसलिए ये Expression भी True या 1 Return करता है। अब यदि हम Logical Operator के उपरोक्त Expression को Represent करें, तो इस Statement को निम्नानुसार Represent कर सकते हैं:

```
X = 1 && 1
```

“C” Language में 0 के अलावा किसी भी संख्या को True ही माना जाता है, फिर चाहे संख्या Positive हो या Negative, इसलिए इस Expression में यदि हम देखें तो Logical AND Operator के दोनों और True या 1 है, अतः ये Logical AND Operator भी True या 1 ही Return करेगा और Variable Identifier X में 1 यानी **True** Store हो जाएगा।

OR (||)

इस Logical Operator के दोनों तरफ की Condition में से यदि किसी एक तरफ की Condition भी **True** होती है, तब भी ये Logical Operator **True** या **1** Return करता है। यदि Logical Operator केवल एक ही स्थिति में **False** Return करता है, जब इस Logical Operator के Left Hand Side व Right Hand Side दोनों तरफ की Conditions **False** होती हैं। जैसे:

```
X = (10 < 5) || (5 < 3)
```

ये Statement Identifier X में 0 यानी False Store करेगा, क्योंकि इस Statement के Execute होने पर सबसे पहले Logical OR Operator के Left Hand Side के Expression (10 < 5) का Execution होता है और ये Expression उस स्थिति में True Return करता है, जब 10 का मान 5 के मान से छोटा होता है।

चूंकि 10 हमेशा ही 5 से बड़ा होता है, इसलिए ये Expression False Return करता है। फिर Logical OR Operator के Right Side की Condition Check होती है, जो उस स्थिति में True या 1 Return करता है, जब 5 का मान 3 से कम होता है। चूंकि यहां भी 5 का मान हमेशा ही 3 से ज्यादा होता है, इसलिए ये Expression भी False या 0 Return करता है। अब यदि हम Logical Operator के उपरोक्त Expression को Represent करें, तो इस Statement को निम्नानुसार Represent कर सकते हैं:

```
X = 0 || 0
```

इस Expression में Logical OR Operator के दोनों और False या 0 है, अतः ये Logical OR Operator False या 0 ही Return करेगा और Variable Identifier X में 0 यानी False Store हो जाएगा।

NOT (!)

ये एक ऐसा Unary Logical Operator है। इस Operator को काम करने के लिए केवल एक ही Operand की जरूरत होती है। जिस Identifier के साथ इस Operator को Use किया जाता है, ये Operator उस Identifier की Condition को Invert कर देता है।

यानी यदि किसी Expression से True Return हो रहा हो, तो इस Operator का प्रयोग करने से वह False Return करने लगेगा और यदि किसी Expression से False Return हो रहा हो, तो उस Expression में इस Operator का प्रयोग करने पर वह Expression True Return करने लगेगा। इस प्रक्रिया को हम निम्नानुसार Expression द्वारा समझ सकते हैं: माना

```
int A = 6;
int B ;
B = !A
```

यदि किसी Program में हम इस Expression को Execute करें और Variable B के मान को Print करें, तो हमें Output में 0 या False प्राप्त होता है। ऐसा इसलिए होता है क्योंकि Variable A में 6 Store है, जो कि एक True मान है, लेकिन जब हम इसके साथ NOT Logical Operator का प्रयोग करके Return होने वाले मान को Variable B में Store करते हैं, तो ये Operator Variable A के True मान को False में Convert कर देता है। इसलिए यदि हम Variable B को Output में Print करते हैं, तो हमें Output में 0 प्राप्त होता है, जो कि False को Represent करता है।

Assignment Operators

किसी भी Program में हमें विभिन्न प्रकार के Identifiers को समय-समय पर विभिन्न प्रकार के मान Initialize या Assign करने की जरूरत पड़ती है। इस जरूरत को पूरा करने के लिए हमें जिस Operator का प्रयोग करना होता है, उसे Assignment Operator कहते हैं।

हालांकि Assignment Operator तो केवल एक ही है, लेकिन इसे कई अन्य तरीकों से भी Use कर सकते हैं। Assignment Operator को जिन अन्य तरीकों से Use किया जाता है, उन तरीकों को Short Hand तरीके कहा जाता है। "C" Language में निम्नानुसार 6 तरीकों से किसी Assignment Operator का प्रयोग किया जा सकता है:

Operator	Declaration	Example	Example Explanation
=	Assignment	A = 10	A = 10
+=	Assigning Sum	A += 10	A = A + 10
-=	Assigning Difference	A -= 10	A = A - 10
*=	Assigning Product	A *= 10	A = A * 10
/=	Assigning Dividend	A /= 10	A = A / 10
%=	Assigning Remainder	A %= 10	A = A % 10

कई बार हमें ऐसी जरूरत होती है, जिसमें किसी एक ही Identifier के मान के साथ किसी प्रकार की प्रक्रिया करने के बाद Generate होने वाले मान को वापस उसी Identifier में Store करना होता है। इस प्रकार का काम करने के लिए हम **Short Hand Assignment Operators** का प्रयोग करते हैं।

उदाहरण के लिए मानलो कि किसी Identifier **A** का मान **10** है और हम चाहते हैं कि इस Identifier में **20** जोड़ कर प्राप्त होने वाले मान **30** को फिर से Identifier **A** में ही Store कर दिया जाए। इस काम को पूरा करने के लिए सामान्यतया हमें निम्नानुसार Statement लिखना होता है:

```
A = A + 20;
```

इसी Statement द्वारा पूरे होने वाले काम को यदि हम और छोटे रूप में लिखना चाहें, तो निम्नानुसार लिख सकते हैं:

```
A += 20;
```

ये Statement भी वही काम करता है, जो पिछला वाला Statement कर रहा है। यानी **A** के मान में 20 जोड़ कर प्राप्त होने वाले मान 30 को फिर से **A** में Store कर देता है। इसी तरह से हम अन्य Assignment Operators का भी प्रयोग कर सकते हैं, जिन्हें उपरोक्त सारणी में उदाहरण के रूप में दर्शाया गया है।

Exercise:

Explain the Short Hand Assignment Operators using appropriate example?

Increment and Decrement Operators

कई बार हमें हमारे Program में क्रम से एक-एक बढ़ने या घटने वाली संख्याओं को Generate करने की जरूरत पड़ती है। इस प्रकार की जरूरत को पूरा करने के लिए हमें Increment (++) या Decrement (--) Operators का प्रयोग करना पड़ता है। वेरियेबल के साथ इनकी दिशा बदल देने से इनके स्वभाव में परिवर्तन आ जाता है।

जब किसी Variable के मान में क्रम से कोई संख्या जोड़ कर वापस उसी Variable में Store कर देते हैं, तो उस Variable का मान उस जोड़ी गई संख्या के अनुसार उसी क्रम में बढ़ता जाता है, इस प्रक्रिया को Variable का **Increment** होना कहते हैं।

उदाहरण के लिए माना एक Variable **x = 0** है और हम चाहते हैं कि इसका मान क्रम से एक-एक बढ़ता जाए। इस जरूरत को पूरा करने के लिए हम निम्नानुसार Statement लिख सकते हैं:

```
x = x + 1
```

हम इसी Statement को $x = x + 1$ लिखने के बजाय संक्षिप्त रूप में **x++** भी लिख सकते हैं।

इसी तरह से जब Variable के मान में से क्रम से कोई संख्या घटा कर प्राप्त मान वापस उसी Variable में Store कर देते हैं, तो इस प्रक्रिया को Variable का **Decrement** होना कहते हैं।

उदाहरण के लिए माना $x = 10$ है व हम क्रम से x का मान 1 कम करना चाहते हैं। इस जरूरत को पूरा करने के लिए हम निम्नानुसार Statement लिख सकते हैं:

```
x = x - 1
```

हम इसी Statement को $x = x - 1$ लिखने के बजाय संक्षिप्त रूप में **x--** भी लिख सकते हैं।

जब हमें किसी Variable के मान को एक-एक के क्रम में ही बढ़ाना या घटाना होता है, या एक-एक के क्रम में ही **Increment** या **Decrement** करना होता है, तब हम जिन दो Operators को Use करते हैं, उन्हें Increment (++) व Decrement (--) Operators कहते हैं। इन दोनों Operators को भी दो-दो तरीकों से Use किया जाता है, जो कि निम्नानुसार हैं:

1 Pre – Increment

जब किसी Variable के पहले Increment ++ का चिन्ह लगाया जाता है, तब उस Variable का मान पहले Increase होता है, उसके बाद वह Variable उस Expression में भाग लेता है, जिसमें उस Variable को Use किया गया है। जैसे

```
int x = 0, y = 10, Result;
Result = ++x + y
```

इस Code Segment में पहले x का मान Increment हो कर 0 से 1 हो जाता है, उसके बाद x का मान 1 y के मान 10 में जुड़ कर 11 Return करता है और Result में 11 Store हो जाता है। अब यदि x , y व Result तीनों को Print किया जाए, तो तीनों का मान क्रमशः 1, 10 व 11 Print होगा।

2 Post – Increment

जब किसी Variable के बाद में Increment चिन्ह लगाया जाता है, तो वह Variable पहले उस Expression में भाग लेता है, जिसमें उसे Use किया गया है, उसके बाद उस Variable का मान Increment होता है। जैसे:

```
int x = 0, y = 10, Result;
Result = x++ + y
```

इस Code Segment में पहले $(x + y)$ Expression Execute होगा और इस Expression से Generate होने वाला Resultant मान 10 Variable Result में Store होगा। उसके बाद x का मान Increment होकर 1 होगा। इस Statement के Execute होने के बाद यदि हम x , y व Result तीनों के मानों को Screen पर Display करें, तो हमें क्रमशः 1, 10 व 10 प्राप्त होगा।

3 Pre – Decrement

Pre-Increment की तरह ही जब किसी Variable के पहले Decrement का चिन्ह लगाया जाता है, तब उस Variable का मान पहले Decrease होता है, उसके बाद वह Variable उस Expression में भाग लेता है, जिसमें उस Variable को Use किया गया है। जैसे

```
int x = 10, y = 20, Result;
Result = --x + y
```

इस Code Segment में पहले x का मान Decrement हो कर 10 से 9 हो जाता है, उसके बाद x का मान 9 y के मान 20 में जुड़ कर 29 Return करता है और Result में 29 Store हो जाता है। अब यदि x, y व Result तीनों को Print किया जाए, तो तीनों का मान क्रमशः 9, 20 व 21 Print होगा।

4 Post – Decrement

Post-Increment की तरह ही जब किसी Variable के बाद में Decrement चिन्ह लगाया जाता है, तो वह Variable पहले उस Expression में भाग लेता है, जिसमें उसे Use किया गया है, उसके बाद उस Variable का मान Decrement होता है। जैसे:

```
int x = 10, y = 20, Result;
Result = x-- + y
```

इस Code Segment में पहले (x + y) Expression Execute होगा और इस Expression से Generate होने वाला Resultant मान 30 Variable Result में Store होगा। उसके बाद x का मान Decrement होकर 9 होगा। इस Statement के Execute होने के बाद यदि हम x, y व Result तीनों के मानों को Screen पर Display करें, तो हमें क्रमशः 9, 20 व 30 प्राप्त होगा।

चलिए, एक Program में इन चारों Operators को Practically Use करके Result देखते हैं। Program निम्नानुसार है:

Program

```
#include <stdio.h>
main()
{
    int x = 10, y = 20, z = 30;
    printf("\n x = 10, y = 20, z = 30 \n");

    printf("\n ++x + y = %d", ++x + y);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
    printf("\n y++ + z = %d", y++ + z);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
    printf("\n --z + x = %d", --z + x);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
    printf("\n y-- + x = %d", y-- + x);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
}
```

Output

```
x = 10, y = 20, z = 30
```

```
++x + y = 31  x = 11, y = 20, z = 30
```

```
y++ + z = 50  x = 11, y = 21, z = 30
```

```
--z + x = 40  x = 11, y = 21, z = 29
```

```
y-- + x = 32  x = 11, y = 20, z = 29
```

Exercise:

Explain the flow of this program?

Bit wise Operators

अभी तक हमने जितने भी Operators के बारे में जाना है, वे सभी Operators किसी Identifier के पूरे Byte पर Operation Perform करते हैं। लेकिन "C" Language में कुछ ऐसे Operators भी Provide करता है, जिनका प्रयोग हम किसी Identifier की Bits पर कर सकते हैं। इस Operator का उपयोग सीधे ही किसी Identifier की Bits पर काम करने के लिए किया जाता है। ये Operator हमेशा Integer प्रकार के Data Type के साथ ही Use होता है, यानी Bitwise Operators को केवल Integer प्रकार के Data Type के Identifier के साथ ही प्रक्रिया करने के लिए Use किया जा सकता है। "C" Language में इनकी कुल संख्या छः होती है:

- & Bitwise **AND** Operator
- ! Bitwise **OR** Operator
- ^ Bitwise **Exclusive OR** Operator
- << Bitwise **SHIFT LEFT** Operator
- >> Bitwise **SHIFT RIGHT** Operator
- ~ Bitwise **Ones Compliment** Operator

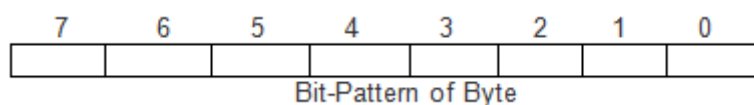
जैसाकि हमने पहले भी कहा है कि Computer की memory में किसी मान को Store किए बिना हम उस मान के साथ किसी प्रकार की कोई प्रक्रिया नहीं कर सकते हैं। लेकिन चूंकि Computer केवल एक Electronic Machine है, इसलिए हम इसमें जिस किसी भी मान को Store करते हैं, वह मान Binary Digits के रूप में ही Store होता है।

जब हम किसी Identifier का प्रयोग करके किसी Memory Location को Access करते हैं, तब वास्तव में हम उस Identifiers से Associated Memory Block के पूरे Byte को Use कर रहे होते हैं। लेकिन "C" हमें कुछ **Bitwise Operator** भी Provide करता है, जिनका प्रयोग करके हम किसी Identifier से Associated पूरे Memory Byte को Access करने के बजाय Memory Byte के किसी भी Bit को Access करने की क्षमता प्राप्त करते हैं। ,

इन Bitwise Operators का यदि ठीक तरह से प्रयोग किया जाए, तो हम हमारे Program की Speed को बहुत तेज कर सकते हैं, क्योंकि Bytes को Access करने की तुलना में Bits को Access करने में Computer को कम समय लगता है। साथ ही इन Operators का प्रयोग करके Directly Memory Location के Bits को Access कर पाने के कारण हम Computer के Hardware को भी Directly Access कर पाने में सक्षम हो सकते हैं।

Bitwise Operators को समझने से पहले हमें ये समझना होगा कि Computer विभिन्न प्रकार के Character व Integer मानों को Computer की Memory में किस तरह से Store करता है। Computer में 8 Bits के समूह को Byte, 16 Bits के समूह को Word तथा 32 Bits के समूह को DWord कहा जाता है।

हम जब भी Character प्रकार का कोई Identifier Create करते हैं, तो Computer एक Byte की Space को Reserve करता है, जो कि 8 Bits का एक समूह होता है। इस Byte को हम निम्नानुसार चित्र द्वारा Represent कर सकते हैं:



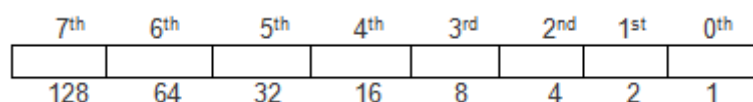
किसी भी Byte या Word के Bits की Position को हमेशा Right Side से Count किया जाता है। इस Byte के हर Bit में केवल **True** या **False** यानी **0** या **1** ही Store हो सकता है। किसी Byte के पहले Bit (0th Position Bit) को **Least Significant Bit** (LSB) व अन्तिम Bit (7th Position Bit) को **Most Significant Bit** (MSB) कहा जाता है।

किसी Signed Character Data Type के Identifier में Most Significant Bit हमेशा Sign को Represent करता है। इस Bit का मान यदि 0 होता है, तो इसमें Stored Bit Pattern किसी Positive संख्या को Represent करता है, जबकि यदि इस Bit का मान 1 हो, तो इसमें Stored Bit Pattern किसी Negative संख्या को Represent करता है।

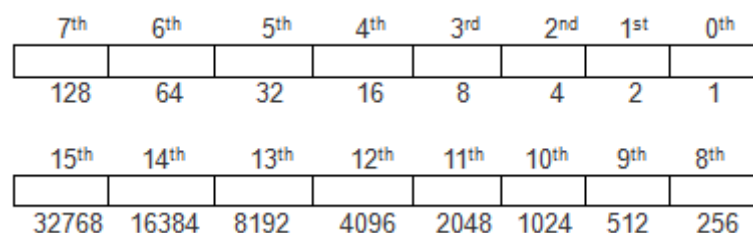
किसी Byte में स्थित हर Bit का उसकी Position के आधार पर एक Unique मान होता है। किसी Byte के Least Significant Bit में Store हो सकने वाली संख्या का अधिकतम मान 1 हो सकता है और हम जैसे-जैसे Least Significant Bit से Most Significant Bit की तरफ बढ़ते जाते हैं, वैसे-वैसे उन Bits के मान Store करने की क्षमता पिछले मान की तुलना में Double होती जाती है।

उदाहरण के लिए किसी भी Byte के 0th Position में Store हो सकने वाला अधिकतम मान 1 हो सकता है, इसलिए 1st Position पर Store हो सकने वाली संख्या का मान पिछले Bit की अधिकतम Capacity से दुगुना, यानी 2 हो सकता है।

इसी तरह से 3rd Position पर Store हो सकने वाली संख्या का अधिकतम मान 4 हो सकता है इसी तरह से आगे भी यही क्रम जारी रहता है। यदि हम इस प्रक्रिया को चित्र द्वारा Represent करें, तो निम्न चित्र में हम देख सकते हैं कि किस प्रकार से हर Bit **LSB** से **MSB** की तरफ बढ़ते हुए दुगुने मान को Represent करने लगता है:



चूंकि Character Type का Variable Memory में 1 Byte या 8 Bit लेता है, इसलिए हमने इस प्रक्रिया को एक Character प्रकार के Identifier पर Apply करके समझाया है। लेकिन ये प्रक्रिया एक Integer प्रकार के Word पर भी पूरी तरह से Apply होती है, अन्तर केवल इतना है कि एक Integer में कम से कम 16 Bits होते हैं, इसलिए Bits की संख्या व उनकी Location के आधार पर एक Integer में Data को Store करने की क्षमता काफी बढ जाती है, जिसे हम निम्न चित्र द्वारा समझ सकते हैं:



इस तरह से यदि किसी Unsigned Character प्रकार के Identifier के सभी Bits में निम्न चित्रानुसार **False** यानी **0** Store हो, तो ये Byte Numerical मान 0 को Represent करता है।

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
0	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1

इसी तरह से यदि इस Byte के सभी Bits का मान **True** या **1** हो, तो ये Byte निम्न चित्रानुसार Numerical मान **255** को Represent करता है।

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

Byte में Stored Binary Bits के आधार पर Decimal Number प्राप्त करने का तरीका ये है कि Byte में जिस-जिस Bit Position पर **True** या **1** Stored होता है, उन्हें आपस में जोड़ लिया जाता है। इस जोड़ से प्राप्त होने वाला मान उस Binary Bit-Pattern का Decimal मान होता है। उदाहरण के लिए निम्न Byte Representation में 0th, 1st व 3rd Bit Position पर True या 1 Stored है, इसलिए इस Binary का Decimal मान $1 + 2 + 8 = 11$ होगा।

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
0	0	0	0	1	0	1	1
128	64	32	16	8	4	2	1

Decimal Value of this Binary [00001011] = $8 + 2 + 1 = 11$

हम जानते हैं कि किसी भी Byte में उसकी Bit Position पर 0 या 1 ही हो सकता है, जहां 0 Lowest Value को Represent करता है, जबकि 1 Highest Values को Represent करता है। इसलिए यदि इस तरीके के आधार पर हम किसी Unsigned Character प्रकार के Identifier के सभी Bits में 0 Store कर दें, तो Byte में Store हो सकने वाला Minimum मान 0 ही हो सकता है, जबकि यदि सभी Bit Position को उनकी High Value यानी 1 से Fill कर दें, तो Byte में Store हो सकने वाला Maximum मान $[1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255]$ ही हो सकता है। यही वजह है कि किसी Unsigned Character प्रकार के Data Type में हम इससे बड़ी संख्या को Store नहीं कर सकते हैं।

जब हम Unsigned Byte में Store हो सकने वाले मान की गणना करते हैं, तब हमें 0 से 255 की Range प्राप्त होती है, लेकिन जब हम एक Byte में Signed प्रकार के मान के Store होने की Range ज्ञात करना चाहते हैं, तब Byte का **Most Significant Bit Sign** की Information को Hold करने लगता है। यदि इस Bit का मान 0 होता है, तो इसमें Stored संख्या Positive होती है, जबकि इस Bit में Stored मान 1 होने की स्थिति में इसमें Stored Bit-Pattern के आधार पर Generate होने वाली संख्या Negative हो जाती है। इस स्थिति में यदि हम किसी Sign वाली संख्या की Minimum व Maximum Range ज्ञात करना चाहें, तो Minimum संख्या का Bit-Pattern निम्नानुसार बनेगा:

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
1	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1

हम देख सकते हैं कि इस Bit Pattern में केवल Most Significant Bit यानी **Sign Bit** ही 1 है जो बता रहा है, कि ये संख्या एक Negative संख्या है, साथ ही हम ये भी देख सकते हैं कि इस **MSB** की Position पर **True** या 1 Store होने का मतलब ये भी है कि इस संख्या का Decimal मान

128 है। यानी इस संख्या का वास्तविक मान **-128** है। अब यदि हम इस Byte के Bit-Pattern को **Invert** कर दें, यानी True को False व False को True कर दें, तो हमें निम्नानुसार Bit-Pattern प्राप्त होता है:

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
0	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

इस Bit-Pattern में **MSB** का मान **0** है जो बताता है, कि इस Byte में Stored संख्या एक **Positive** संख्या है। अब यदि हम इस संख्या का **Decimal** मान ज्ञात करें, तो हमें [1 + 2 + 4 + 8 + 16 + 32 + 64 = 127] प्राप्त होता है, जो कि इस Byte में Store हो सकने वाला **Maximum** मान है।

जिस तरह से हमने एक Byte की Bit Position पर Stored **0** या **1** के आधार पर एक Byte में Store हो सकने वाले न्यूनतम व अधिकतम मान को **Calculate** किया है, उसी तरह से हम **Integer** व **Long Integer** Type के Identifier में Store हो सकने वाले न्यूनतम व अधिकतम मान को भी ज्ञात कर सकते हैं।

इसके लिए हमें केवल ये ध्यान रखना होता है, कि 2 Byte के Integer में 16 Bit होते हैं, इसलिए 2 Byte के Integer की Limit ज्ञात करने के लिए हमें 16 Bits के Pattern के आधार पर Range ज्ञात करना पड़ेगा। जबकि Long Integer प्रकार का Identifier Memory में 4 Byte या 32 Bit Reserve करता है, इसलिए Long Integer प्रकार के Identifier में Store हो सकने वाली **Minimum** व **Maximum** संख्या का मान प्राप्त करने के लिए हमें 32 Bit के Pattern बनाने पड़ेंगे।

एक बात हमें ध्यान रखें, कि जब किसी Identifier में Store होने वाली संख्या **Unsigned** होती है, तब Byte, Word व DWord के सभी Bits के आधार पर उसमें Store हो सकने वाली **Maximum** व **Minimum** संख्या तय करते हैं, लेकिन जब किसी Byte, Word या DWord में Store होने वाली संख्या **Sign** वाली होती है, तब Byte, Word व DWord के सभी Bits मिलकर उनमें Store होने वाली संख्या की **Range** तय नहीं करते हैं, बल्कि इन सभी का **MSB** संख्या का **Sign** तय करने का काम करते हैं।

इस कारण से एक **Unsigned Identifier** जितना मान Store कर सकता है, **Sign** वाला Identifier उसका आधा ही Store करता है। यानी Sign वाले Identifier की **Range Positive** संख्या Store करने के लिए आधी कम हो जाती है, लेकिन **Negative** संख्या Store करने के लिए आधी बढ़ जाती है।

दसमलव वाले मान भी Computer में Bit-Pattern के रूप में ही Store होते हैं, लेकिन उनके Store होने के तरीके में थोड़ा अन्तर होता है और अन्तर ये होता है कि हर दसमलव वाली संख्या **Sign** वाली संख्या ही होती है। यानी **Float**, **Double** या **Long Double** संख्या कभी भी **Unsigned** नहीं होती है। साथ ही इनके Bit-Pattern के कुछ Bits दसमलव के बाद वाली संख्या को **Represent** करने का काम करते हैं, जबकि कुछ Bits दसमलव से पहले वाली संख्या को **Represent** करने का काम करते हैं।

जिस तरह से हम किसी Byte में Stored Bit-Pattern के आधार पर **Decimal** संख्या ज्ञात कर सकते हैं, उसी तरह से हम किसी **Decimal** संख्या का **Bit Pattern** भी बना सकते हैं। उदाहरण के लिए मानलो हमें मान **57** का Bit-Pattern ज्ञात करना है। इस Bit-Pattern को ज्ञात करने के लिए हमें निम्न क्रम को **Use** करना होता है:

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
128	64	32	16	8	4	2	1

- 1 सबसे पहले हमें ये पता लगाना होता है, कि हम जिस संख्या का Bit-Pattern ज्ञात करना चाहते हैं, उस संख्या को Represent करने वाला कोई Bit Byte के Bit-Pattern में उपलब्ध है या नहीं। यदि ज्ञात की जाने वाली संख्या का मान Bit-Pattern में ना हो, तो ज्ञात की जाने वाली संख्या से छोटी संख्या के Bit को **True** या **1** कर देना चाहिए।

उपरोक्त चित्र में हम देख सकते हैं कि इस Byte के Bit-Pattern में 1, 2, 4, 8, 16, 32, 64 व 128 हैं, लेकिन **57** नहीं है। इस स्थिति में 57 से Just छोटा मान 32 है, इसलिए हमें Byte के Bit Pattern में इसी मान को True या 1 करना होता है, जैसाकि हमने निम्न चित्र में किया है।

		1					
7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
128	64	32	16	8	4	2	1

- 2 अब ज्ञात की जाने वाली संख्या के मान में से Set किए गए Bit की संख्या को घटाना होता है।

चूंकि हमने 32 के मान के Bit को Set किया है, इसलिए हमें 57 में से 32 को घटाना होता है। 57 में से 32 को घटाने पर 25 बचता है, इसलिए अब हमें इस 25 की Binary Set करनी है। इसके लिए हमें फिर से पिछले Step को Use करना होता है।

चूंकि हमारे Byte के Pattern में मान 25 के लिए भी कोई संख्या नहीं है, इसलिए हमें 25 से छोटे मान के Bit को Set करना होता है, जो कि हमारे इस उदाहरण में 16 है। इस Bit को 1 कर देने पर हमें निम्नानुसार Pattern प्राप्त होता है:

		1	1				
7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
128	64	32	16	8	4	2	1

- 3 अब हमें फिर से बची हुई संख्या को ज्ञात करना होता है। इसके लिए हमें Current संख्या में से Set की गई Bit की संख्या को घटाना होता है।

चूंकि हमारी Current संख्या 25 है, जिसमें से हमने 16 को Set किया है, इसलिए अब हमें 25 में से 16 को घटाना होता है। ऐसा करने पर हमें Resultant मान $25 - 16 = 9$ प्राप्त होता है, जिसके लिए हमें Bit को Set करना होता है।

यहां फिर हमें Step1 को Use करना होता है, जिससे हमें मान 8 मिलता है, जिसे Set करना होता है। क्योंकि मान 8 ही मान 9 से सबसे कम छोटा मान है। मान 8 के Bit को Set करने पर हमें निम्नानुसार Bit-Pattern प्राप्त होता है:

		1	1	1			
7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
128	64	32	16	8	4	2	1

- 4 Current मान 9 में से Currently Set किए गए Bit के मान 8 को घटाने पर हमें 1 प्राप्त होता है, और अब हमें केवल मान 1 के लिए Bit को Set करना है। चूंकि मान 1 को Represent करने वाला bit 0th Position पर है, इसलिए केवल इस Bit को Set कर देने पर हमें 57 का Binary Bit-Pattern प्राप्त हो जाएगा, जो कि निम्नानुसार है:

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
		1	1	1			1
128	64	32	16	8	4	2	1

चूंकि हमें हमारे Required मान की Binary Bit-Pattern प्राप्त हो चुकी है, इसलिए जिन भी Bits की Position Blank है, उनमें 0 Fill कर देने से हमें हमारे Required मान की Actual Bit-Pattern प्राप्त हो जाती है, जो कि निम्नानुसार है:

7 th	6 th	5 th	4 th	3 rd	2 nd	1 st	0 th
0	0	1	1	1	0	0	1
128	64	32	16	8	4	2	1

हमें मान 57 का Binary Bit Pattern **00111001** प्राप्त हुआ है। ये Bit Pattern सही है या नहीं इस बात की जांच करने के लिए हम उन Bits के मानों को जोड़ सकते हैं, जिनमें **True** या **1** Stored है। हमारे Bit-Pattern में 0th, 3rd, 4th व 5th Position के Bits **On** है, जिनके मानों का Total $[1 + 8 + 16 + 32 = 57]$ है, जो कि वही मान है, जिसका Bit-Pattern हम बनाना चाहते थे, इसलिए हमारा Bit-Pattern सही है।

इस तरह से हम किसी भी Bit-Pattern का Decimal मान ज्ञात कर सकते हैं और किसी भी Decimal संख्या का Bit Pattern बना सकते हैं।

Bitwise AND Operator (&)

ये Operator Use करके हम दो Identifier के Bits पर AND Masking की प्रक्रिया को Apply करते हैं। AND Masking में दोनों Identifiers के Bits आपस में AND Form में Compare होते हैं। यदि दोनों Identifiers में समान Position पर Bit का मान 1 हो यानी Bit **True** हो तो Resultant Bit भी True होता है, अन्यथा Resultant Bit False हो जाता है। उदाहरण के लिए **19** की Binary 10011 होती है और **21** की Binary 10101 होती है। अब यदि निम्नानुसार दो Variables में ये दोनों मान Stored हों:

```
int firstValue = 19;    //Binary : 10011
int secondValue = 21;  //Binary : 10101
int resultValue;
```

और यदि इन दोनों Identifiers पर निम्नानुसार AND Masking करके Resultant मान को result नाम के Variable में Store किया जाए, तो Result निम्नानुसार प्राप्त होगा :

```
result = firstValue & secondValue;

firstValue's Binary   :   10011           // Decimal Value = 19
secondValue's Binary  :   10101           // Decimal Value = 21
-----
resultValue's Binary   :   10001           // Decimal Value = 17
-----
```

AND Masking में निम्नानुसार Table के अनुसार Bits पर प्रक्रिया होती है, जिसमें यदि दोनों Identifiers के समान Position के दोनों Bits का Comparison होता है और समान Position पर ही Resultant Bit Return होता है।

AND Mask	0	1
0	0	0
1	0	1

इस Bitwise Operator का प्रयोग अक्सर ये जानने के लिए किया जाता है, कि किसी Operand का कोई अमुक Bit ON (1) है या OFF (0) किसी Operand का कोई Bit On है या Off, ये जानने के लिए हमें एक अन्य Operand लेना होता है और उस Operand में उस Bit को On रखा जाता है, जिसे प्रथम Operand में Check करना होता है।

उदाहरण के लिए माना एक Operand का Bit Pattern 11000111 है और हम जानना चाहते हैं कि इस Pattern में चौथा Bit ON है या नहीं। ये जानने के लिए हमें एक दूसरा Operand लेना होगा और उस Operand के Bit Pattern में चौथे Bit को ON(1) व शेष Bits को OFF(0) रखना होगा। इस प्रकार से हमें दूसरे Operand का जो Bit Pattern प्राप्त होगा वह 00001000 होगा, जिसका चौथा Bit ON है।

किसी Operand के Bit Pattern के किसी Bit की स्थिति पता करने के लिए दूसरा Bit Pattern लेकर जो प्रक्रिया की जाती है, उसे **Masking** कहते हैं और जब इस प्रक्रिया में Bitwise Operator & का प्रयोग किया जाता है, तब इसे **AND Mask** कहते हैं।

Trick ये है कि जब हम प्रथम Operand को & Operator द्वारा दूसरे Operand के Bit Pattern से Compare करते हैं तब यदि प्रथम Bit Pattern में चौथा Bit ON होता है, तो ही Comparison से प्राप्त Resultant Bit Pattern में भी चौथा Bit ON होता है अन्यथा चौथा Bit Off होता है। इस Masking को हम निम्नानुसार Represent कर सकते हैं:

```

First Operand   : 11000111
Second Operand  : 00001000
-----
AND MASK        : 00000000
-----

```

इस उदाहरण में हम देख सकते हैं कि पहले Bit-Pattern का चौथा Bit Off है, इसलिए Resultant Bit-Pattern में भी चौथा Bit Off है। अब निम्न Fragments को देखिए:

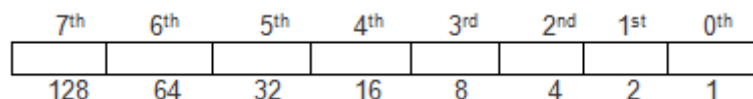
```

First Operand   : 11001001
Second Operand  : 00001000
-----
AND MASK        : 00001000
-----

```

इस Fragment में हम देख सकते हैं कि पहले Bit-Pattern का चौथा Bit On है और यही जानने के लिए कि पहले Operand का चौथा Bit On है या नहीं, हमने एक **Mask Bit Pattern** Create किया है, जिसके चौथे Bit को On रखा है। इस स्थिति में Resultant Bit-Pattern का चौथा Bit केवल उसी स्थिति में On हो सकता है, जब Check किए जा रहे Operand के Bit-Pattern में चौथा Bit On हो। इस तरह से AND Masking के उपयोग द्वारा हमें पहले Operand के चौथे Bit की स्थिति का पता चल जाता है।

किसी भी Bit-Pattern में हर Bit की Position का एक मान होता है। इस Position के मान द्वारा हम Directly उस Bit को Refer कर सकते हैं। उदाहरण के लिए निम्न चित्र को देखिए:



इस चित्र में हर Bit Position के साथ एक Number Associated है। यदि हम किसी Bit-Pattern के चौथे Bit को Refer करना चाहते हैं, तो हमें मान 16 को Use करना होता है। इसी तरह से यदि हमें किसी Bit Pattern के छठे Bit को Access करना हो, तो हमें इस Bit Position से Associated मान 64 को Use करना होता है।

चलिए, अब हम एक उदाहरण द्वारा Logical AND Operator को Use करके किसी Identifier के किसी Particular Bit की Status को Check करते हैं कि वह Bit On है या नहीं।

इस उदाहरण में हमने एक Identifier **x** में एक मान 150 Store किया है, जिसका Bit-Pattern **1001011** होता है। हम इस Bit-Pattern के पांचवे व छठे Position के Bit की ON/OFF Status जानना चाहते हैं। चूंकि हम किसी भी मान के Bit-Pattern को Binary Form में Use नहीं कर सकते हैं, इसलिए किसी Bit Position को Refer करने के लिए हमें उसके साथ Associated Decimal Number को Use करना होता है।

Program

```
#include <stdio.h>

main()
{
    int x = 150;           // Bit-Pattern of 150 = 10010110
    int j;
    clrscr();

    printf("\n Value of x is %d ", x);

    j = x & 16;
    (j == 0) ? printf("\n Fifth Bit of value %d is Off", x) :
    printf("\n Fifth Bit of value %d is On", x);

    j = x & 32;
    (j == 0) ? printf("\n Sixth Bit of value %d is Off", x) :
    printf("\n Sixth Bit of value %d is On", x);
}
```

Output

```
Value of x is 150
Fifth Bit of value 150 is On
Sixth Bit of value 150 is Off
```

जब ये Program Run होता है, तब निम्नानुसार Form में $j = x \& 16$; व $j = x \& 32$; Statements को Execute करता है:

For Fifth Bit of the value of x: **$j = x \& 16;$**

Bit-Pattern of x	:	10010110
Bit-Pattern of Mask	:	00010000

AND MASK	:	00010000

For Sixth Bit of the value of x: **$j = x \& 32;$**

Bit-Pattern of x	:	10010110
Bit-Pattern of Mask	:	00100000

AND MASK	:	00000000

चूंकि जब हम पांचवे Bit को Check करते हैं, तब मान 150 का पांचवा Bit On होने की वजह से Making Process से 1 Generate होता है और ये 1 Variable j में Store हो जाता है। फिर Ternary Operator में (j==0) Expression Execute होता है, जो कि False हो जाता है, क्योंकि j का मान 1 है और (1==0) नहीं होता है। इस वजह से Ternary Operator के दूसरे Statement का Execution हो जाता है, जो Output में निम्नानुसार Message प्रदान करता है:

Fifth Bit of value 150 is On

लेकिन जब हम छठे Bit को Check करते हैं, तब मान 150 का छठा Bit Off होने की वजह से Masking Process से 0 Generate होता है और ये 0 Variable j में Store हो जाता है। अगले Statement में फिर से (j==0) Expression Execute होता है, जो इस बार True होने की वजह से Ternary Statement के पहले Statement का Execution कर देता है और हमें निम्नानुसार Output प्राप्त होता है:

Sixth Bit of value 150 is Off

इस प्रकार से हम किसी भी Identifier के मान के किसी Particular Bit को On/Off Status की जानकारी प्राप्त करने के लिए Bitwise AND Operator का प्रयोग कर सकते हैं। सामान्यतया विभिन्न प्रकार के Bitwise Operators का प्रयोग विभिन्न प्रकार के Hardware Devices के साथ प्रक्रिया करने के लिए ही करते हैं।

Bitwise OR Operator (|)

ये Operator Use करके हम दो Identifier के Bits पर OR Masking की प्रक्रिया को Apply करते हैं। OR Masking में दोनों Identifiers के Bits आपस में OR Form में Compare होते हैं। यदि दोनों Identifiers में से किसी एक भी Identifier में समान Position पर Bit का मान 1 हो यानी Bit True हो तो Resultant Bit भी True होता है, अन्यथा Resultant Bit False हो जाता है। OR Masking को हम निम्नानुसार समझ सकते हैं, जहां दो Variables में 19 व 21 मान Stored है:

```
int firstValue = 19;    //Binary : 10011
int secondValue = 21;  //Binary : 10101
int resultValue;
```


इन दोनों Identifiers पर OR Masking की प्रक्रिया को Apply करने पर निम्नानुसार Operation Perform होते हैं:

```
result = firstValue | secondValue;

firstValue's Binary      :   10011          // Decimal Value = 19
secondValue's Binary     :   10101          // Decimal Value = 21
-----
resultValue's Binary     :   10111          // Decimal Value = 23
-----
```

OR Masking में Identifier के Bits पर निम्नानुसार Table के अनुसार पर प्रक्रिया होती है, जिसमें दोनों Identifiers के समान Position के दोनों Bits का आपस में Comparison होता है और तीसरे Identifier में समान Position पर ही Resultant Bit Return होता है।

OR Mask	0	1
0	0	1
1	1	1

Bitwise OR Operator का प्रयोग किसी Bit Pattern में स्थित किसी खास Bit को ON करने के लिए किया जाता है। जब हमें किसी अमुक Bit को किसी Bit-Pattern में ON करना होता है, तब हमें एक और Bit-Pattern की जरूरत होती है। इस दूसरे Bit-Pattern को **OR Mask Bit Pattern** कहते हैं। इस OR Mask में हमें केवल उसी Bit का मान **1** रखना होता है, जिसे हम हमारे प्रथम Bit-Pattern में **ON** करना चाहते हैं।

चलिए, एक उदाहरण द्वारा OR Masking की प्रक्रिया को समझते हैं। मानलो कि हम Bit-Pattern 10100110 (Value = **150**) की चौथी Bit को On करना चाहते हैं। इस जरूरत को पूरा करने के लिए हमें OR Mask के रूप में Bit-Pattern **00001000** को Use करना होगा। जब हमें इस पर OR Masking की प्रक्रिया करनी हो, तो ये प्रक्रिया निम्नानुसार होगी:

```
result = firstValue | secondValue;

firstValue's Binary      :   10010110        // Decimal Value = 150
secondValue's Binary     :   00001000
-----
resultValue's Binary     :   10011110        // Decimal Value = 158
-----
```

Resultant मान में हम देख सकते हैं, कि इसके केवल चौथे Bit का मान ही 0 से 1 हुआ है। चूंकि चौथे Bit-Position का मान 8 होता है, इसलिए Resultant मान 158 प्राप्त हो रहा है, जो कि Original मान 150 से केवल 8 ही ज्यादा है। इस समस्या का Program निम्नानुसार है:

Program

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

```

{
    int x = 150, j;
    clrscr();
    printf("\n Value of x is %d ", x);

    j = x | 8;
    printf("\n Forth Bit of value %d is Now On", x);
    printf("\n Now the Value of x is %d ", j);

    getch();
}

```

Output:

```

Value of x is 150
Forth Bit of value 150 is Now On
Now the Value of x is 158

```

Bitwise XOR (Exclusive OR) Operator (|)

ये Operator Use करके हम दो Identifier के Bits पर XOR Masking की प्रक्रिया को Apply करते हैं। XOR Masking में दोनों Identifiers के Bits आपस में XOR Form में Compare होते हैं। यदि दोनों Identifiers में से किसी एक भी Identifier में समान Position पर Bit का मान 1 हो यानी Bit **True** हो तो Resultant Bit भी True होता है, लेकिन यदि दोनों ही Identifiers में समान Position पर समान Bit हो, तो Resultant Bit **False** हो जाता है। XOR Masking को समझने के लिए हम पिछले उदाहरण को ही Use कर रहे हैं, जिसमें दो Variables में मान **19** व **21** Stored हैं:

```

int firstValue = 19;    //Binary : 10011
int secondValue = 21;  //Binary : 10101
int resultValue;

```

इन दोनों Identifiers पर XOR Masking की प्रक्रिया को Apply करने पर निम्नानुसार Operation Perform होते हैं:

```

result = firstValue ^ secondValue;

firstValue's Binary   :   10011           // Decimal Value = 19
secondValue's Binary  :   10101           // Decimal Value = 21
-----
resultValue's Binary  :   00110           // Decimal Value = 6
-----

```

XOR Masking में निम्नानुसार Table के अनुसार Bits पर प्रक्रिया होती है, जिसमें दोनों Identifiers के समान Position के दोनों Bits का आपस में Comparison होता है और समान Position पर ही Resultant Bit Return होता है।

XOR Mask	0	1
0	0	1
1	1	0

इस Operator का प्रयोग करके हम किसी Identifier की Bits को Toggle तरीके से बार-बार On/Off कर सकते हैं। यानी जब हमें किसी Identifier के Bits को Toggle तरीके से On/Off करना होता है, तब हम इस Bitwise Operator का प्रयोग करते हैं। इस Operator को हम निम्न Program के अनुसार Use कर सकते हैं:

Program

```
#include <stdio.h>

main()
{
    int x = 50, k=10;
    clrscr();
    printf("\n Value of x is %d ", x);
    printf("\n\n Value of k is %d ", k);

    printf("\n\n k = %d is Masking the value of x = %d \n", k, x);
    x = x ^ k;
    printf("\n After XOR Masking the Value of x is %d \n", x);

    printf("\n k = %d is Masking the Changed Value of x = %d again", k, x);
    x = x ^ k;
    printf("\n\n Now the Value of x is changed again to %d ", x);
}
```

Output

```
Value of x is 50 // Bit-Pattern : 00110010
Value of k is 10 // Bit-Pattern : 00001010
k = 10 is Masking the value of x = 50
After XOR Masking the Value of x is 56 // Resultant : 00111000
k = 10 is Masking the Changed Value of x = 56 again
Now the Value of x is changed again to 50 // Resultant : 00110010
```

One's Complement Bitwise Operator (~)

इस Operator का प्रयोग करके हम किसी भी Identifier के मान की Bits को Invert कर सकते हैं। जब किसी मान की Bits को Invert कर दिया जाता है, तब Generate होने वाले मान का चिन्ह बदल जाता है। इस प्रक्रिया को हम निम्नानुसार समझ सकते हैं:

```
x = 150 //Bit-Pattern : 10010110 = 150
~x //Bit-Pattern : 01101001 = -151
```

One's Complement को समझने के लिए हम निम्नानुसार एक Program बना सकते हैं:

Program

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

main()
{
    int j = 150, k;
    clrscr();
    k = ~j;
    printf("\n Original Value is  = %d", j);
    printf("\n Complemented Value is = %d", k);
    getch();
}
```

Output:

```
Original Value is  = 150
Complemented Value is = -151
```

Right Shift Operator (>>)

ये Operator, Operand के Bits को Right में Shift करने का काम करता है। हमें किसी Operand के Bits को जितना Shift करना होता है, हम इस Operator के बाद वह संख्या लिख देते हैं। जैसे val का मान 128 है और हमें इसके Bits orientation को 2 अंक Right में Shift करना हो तो हम val >> 2 लिखते हैं। इस Statement से val में Stored Bits 10000000 दो Bit Right में Shift हो जाता है और हमें 00100000 प्राप्त होता है। हम जितने Bits Right में Shift करते हैं, Bit-Pattern में Left side में उतने ही 0 fill हो जाते हैं। इसे एक उदाहरण द्वारा देखते हैं।

Program

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

main()
{
    int k = 1028, l;
    clrscr();
    printf("\n Value of Identifier K is %d \n", k);
    l = k >> 2;
    printf("\n After 2-Bits Right Shifting \n");
    printf(" The Value of K is %d \n", l);
    getch();
}
```

Output

```
Value of Identifier K is 1028
After 2-Bits Right Shifting
The Value of K is 257
```

इस प्रोग्राम में हम देख सकते हैं कि Operand k में Stored Bits, Right में Shift हो रहे हैं। जैसे-जैसे Bits Right में Shift होते हैं तो k का मान भी बदलता जाता है। Right Shifting से एक और बहुत महत्वपूर्ण तथ्य सामने आता है, जो ये है कि यदि हम क्रम से किसी संख्या को एक-एक Bit Right Shift करते जाते हैं, तो Operand का मान भी क्रम से आधा होता जाता है।

यानी Right Shifting से हम जितने Bits Right में Shift करते हैं, उतनी ही बार Operand का मान आधा हो जाता है। जैसे इस Program में हुआ है। 1028 को यदि 4 Bit Right में Shift किया जाए तो ये कहा जा सकता है कि 1028 में चार बार दो का भाग दिया गया है। यानी

$$\begin{aligned} 1028/2 &= 514 \\ 514/2 &= 257 \end{aligned}$$

हमने जैसा कि पहले बताया कि हम Bitwise Operators का प्रयोग केवल char या int प्रकार के Operand के साथ ही कर सकते हैं और यहां int प्रकार के Operand k के साथ प्रक्रिया की है। यहां ये सवाल दिमाग में आ सकता है कि $257/2 = 128.5$ होना चाहिये था फिर 256 क्यों हुआ। इसकी वजह यही है कि int प्रकार का मान पूर्णांक में ही हो सकता है। int में दसमलव संख्याएं मान्य नहीं हैं और Bitwise Operators float या double को मान्य नहीं करते, वे केवल int या char को मान्य करते हैं। इसलिए यहां 128.5 ना हो कर 128 ही हुआ है।

Left Shift Operator (<<)

Left Shift Operator के काम करने का तरीका बिल्कुल वही है जो Right Shift Operator का है। लेकिन दोनों के काम करने का क्रम बिल्कुल विपरीत है। ये किसी Operand के Bits को Left में Shift करता है और Right में खाली हुए स्थान को 0 से भर देता है। इसे समझने के लिए हम ऊपर के ही उदाहरण में केवल इतना बदलाव कर रहे हैं, यानी जहां पर Right Shift Operator का प्रयोग किया था, वहां पर Left Shift Operator का प्रयोग कर रहे हैं और k का मान 1028 से बदल कर 128 कर रहे हैं।

Program

```
#include <stdio.h>
main()
{
    int k = 128, l;
    clrscr();

    printf("\n Value of Identifier K is %d \n", k);
    l = k << 2;
    printf("\n After 2-Bits Right Shifting \n");
    printf(" The Value of K is %d \n", l);
}
```

Output

Value of Identifier K is 128

After 2-Bits Right Shifting

The Value of K is 512

जैसाकि हमने अभी बताया कि Left Shift Operator, Right Shift Operator से विपरीत Output देता है। जिस प्रकार से Right Shifting में Operand का मान आधा होता जाता है, उसी प्रकार से

Left Shifting में Operand का मान पूर्व मान से दुगुना होता जाता है। यही वजह है कि मान 128 को 2 Bit Left Shift करने से उसका मान 512 हो गया है। किसी Operand के 2 Bits को Left में Shift करने का मतलब है, उस संख्या को दो बार दुगुना करना। इस प्रक्रिया को हम निम्नानुसार समझ सकते हैं:

```
128 * 2 = 256
256 * 2 = 512
```

Comments

“C” Language में Program लिखते समय विभिन्न प्रकार के Comments दिए जा सकते हैं। ये Comments Programmer अपनी सुविधा के लिए लिखता है। विभिन्न प्रकार के Comments द्वारा एक Programmer Program के Flow को तथा Program में Use किए जाने वाले Special Tricks को Specify करता है, जिससे Program Readable हो जाता है। सामान्यतया Comments को Program के Documentation Section में लिखा जाता है, लेकिन एक Programmer Program में किसी भी स्थान पर Comments लिख सकता है।

“C” Language Program में Comments को लिखने के लिए /* ... */ का प्रयोग किया जाता है। इस Symbol के बीच लिखे जाने वाले Statements केवल Source File में ही उपयोगी होते हैं। Comments कभी भी Compile नहीं होते हैं। Compiler किसी Source File में लिखे गए विभिन्न Comments को Compilation के समय हमेशा Ignore कर देता है, इसलिए Comments की वजह से कभी भी Executable File की Size में कोई फर्क नहीं पड़ता है।

हम एक Program में किसी भी स्थान पर Comment लिख सकते हैं। लेकिन किसी एक Comment के अन्दर दूसरे Comment की Nesting नहीं कर सकते हैं। जैसे

```
/* This is my first C Program */
```

ये एक सामान्य Comment है। लेकिन

```
/* This is my /*first*/ C Program */
```

ये एक गलत Comment है, क्योंकि इसमें एक Comment के अन्दर दूसरे Comment को Nest किया गया है। हम printf() या scanf() जैसे किसी Function में भी Comment को नहीं लिख सकते हैं। यदि हम ऐसा करते हैं, तो Program तो Compile होता है, लेकिन Output में वह Comment भी Print हो जाता है। जैसे:

```
printf("/*This is my first printf() function */ Hello");
```

Output:

```
/*This is my first printf() function */ Hello
```

हम देख सकते हैं कि इस Statement के Output में Comment भी Compile हो रहा है।

Exercise:

- 1 Comments से आप क्या समझते हैं? किसी Program में इसका प्रयोग क्यों किया जाना चाहिए? यदि इसका प्रयोग ना किया जाए, तो Program पर क्या असर पड़ेगा?
- 2 किसी Integer संख्या का Equivalent Binary Bit-Pattern ज्ञात करने का Program बनाओ।
- 3 निम्न Expressions क्या Result Generate करेंगे, जहां A = 10, B = 20 व C = 30 हैं:
I A != 10 && B > 30 II 10 <= C && !(A) == B
III C || A != B IV (A && B) == (B && C)
V C = A++ + ++ B VI C = --A - --B + ++C -C++
VII A += A++ + ++A VIII B /= (A * ++B) - --C - B-
- 4 Left Shift व Right Shift Operator के अन्तर को समझाईए। इन्हें किस परिस्थिति में Use करना चाहिए।
- 5 Logical AND/OR तथा Bitwise AND/OR Operators एक दूसरे से किस प्रकार भिन्न हैं?
- 6 Logical NOT Operator व One's Complement Operator के काम करने के तरीके को समझाईए। इन्हें एक दूसरे के स्थान पर Use करने के लिए हमें Program में किस प्रकार का Change करना पड़ता है? एक उदाहरण द्वारा समझाईए।
- 7 Increment/Decrement Operators को समझाईए। Pre व Post के अन्तर को उदाहरण द्वारा स्पष्ट कीजिए।
- 8 "C" का Compiler किसी भी Non-Zero मान को True मानता है। ये तथ्य सही है या गलत?

How to Get this Ebook in PDF Format

ये पुस्तक केवल **PDF Format Ebook** के रूप में ही Available है और आप इस पुस्तक को केवल हमारी **Official Website** (<http://www.bccfalna.com/>) से ही खरीद सकते हैं। इसलिए यदि आपको ये पुस्तक पसन्द आ रही है और आप इसे PDF Format Ebook के रूप में खरीदना चाहते हों, तो आप इस पुस्तक को Online खरीदने के लिए निम्नानुसार दिए गए **3 Simple Steps Follow** कर सकते हैं:

Select Purchasing EBooks

सबसे पहले <http://www.bccfalna.com/how-to-pay/> Link पर Click कीजिए। जैसे ही आप इस Link पर Click करेंगे, आप हमारी Website के निम्नानुसार **Order Page** पर पहुंच जाएंगे :

<input checked="" type="checkbox"/>	C Programming Language in Hindi	300/-
<input type="checkbox"/>	C++ Programming Language in Hindi	300/-
<input type="checkbox"/>	Java Programming Language in Hindi	300/-
<input checked="" type="checkbox"/>	C# Programming Language in Hindi	400/-
<input type="checkbox"/>	Data Structure and Algorithms in Hindi	250/-
<input type="checkbox"/>	Oracle 8i/9i – SQL/PLSQL in Hindi	300/-
<input type="checkbox"/>	Visual Basic 6 in Hindi	250/-
<input type="checkbox"/>	HTML5 with CSS3 in Hindi	350/-
<input type="checkbox"/>	Advance JavaScript in Hindi	350/-
<input type="checkbox"/>	jQuery in Hindi	350/-
<input type="checkbox"/>	Core PHP in Hindi	350/-
	Total	Rs. 700/- Only.
	Discounted Amount	Rs. 100/- Only.
	Total Payable Amount	Rs. 600/- Only.

इस Page पर आपको उन पुस्तकों को Select करना है, जिन्हें आप खरीदना चाहते हैं। आप जैसे-जैसे पुस्तकें Select करते जाएंगे, आपको उनका **Total Amount**, **Discount** व **Total Payable Amount** उपरोक्त चित्रानुसार दिखाई देने लगेगा, जहां Total Payable Amount ही वह Amount है, जो आपको अपनी Selected EBooks को खरीदने के लिए **Pay** करना होगा।

पुस्तकें Select करने के बाद इसी Page पर दिखाई देने वाले **“Order Details”** Form में आपको निम्न चित्रानुसार अपना **Name**, **Email Address** व **Mobile Number** Specify करके **“Order Now”** पर Click करते हुए उपरोक्त Selected EBooks का Order Place करना होगा:

Order Details	
Your Name	<input type="text" value="Kuldeep Mishra"/>
Email Address	<input type="text" value="bccfalna@gmail.com"/>
Mobile Number	<input type="text" value="09799455505"/>
<input type="button" value="Order Now"/>	

चूंकि ये सारी पुस्तकें Physical Books नहीं बल्कि **PDF Format Ebooks** हैं। इसलिए ये पुस्तकें आपको आपके Email पर ही भेजी जाएंगी, जिन्हें आप अपने Email के माध्यम से अपने Computer पर Download करके अपने PDF Supported *Computer, Mobile, Smart Phone, Tablet PC, Net-Book, Notebook* या *Laptop* जैसी किसी भी Device के माध्यम से पढ़ सकते हैं अथवा यदि आप चाहें, तो अपने Printer द्वारा इन पुस्तकों का Hard Copy Printout निकाल सकते हैं।

इसलिए जरूरी है कि उपरोक्त “**Order Details**” Form पर आप जो **Email Address** व **Mobile Number** Specify करते हैं, वह Working और एकदम सही हो। क्योंकि किसी भी तरह की परेशानी की स्थिति में हम आपको आपके **Mobile Number** पर ही Contact करते हैं।

Pay “Total Payable Amount”

जैसे ही आप “**Order Now**” Button पर Click करेंगे, आपको एक Email मिलेगा, जिसमें आप द्वारा Order की गई EBooks की Details होगी। Selected पुस्तकों का Order Place करने के बाद अब आपको “**Total Payable Amount**” का Payment करना होगा।

यदि आपके पास **Net-Banking** या **Mobile-Banking** की सुविधा है, तो आप Payment करने के लिए अपने Account में Login करके निम्न में से किसी भी Bank A/c में Payment Deposit कर सकते हैं:

**भारतीय स्टेट बैंक**
State Bank of India
With you - all the way

SBI Bank A/c no.	:	31154882587
Account Name	:	Namita Mishra
Branch Name	:	Falna
Address	:	Near Railway Crossing, Falna Station – 306116
IFSC Code	:	SBIN0007868

**बैंक ऑफ़ बड़ौदा**
Bank of Baroda
India's International Bank

BOB Bank A/c no.	:	35260100003212
Account Name	:	Namita Sharma
Branch Name	:	Falna
Address	:	Sanderao Road, Falna, Dist. Pali (Raj.)- Pin-306116
IFSC Code	:	BARB0FALNAX

**स्टेट बैंक ऑफ़ बीकानेर एण्ड जयपुर**
State Bank of Bikaner and Jaipur
The Bank with a vision

SBBJ Bank A/c no.	:	61089986732
Account Name	:	Kuldeep Chand Mishra
Branch Name	:	Bali
Address	:	Sr. Secondary School Road, Bali- 306701
IFSC Code	:	SBBJ0010193

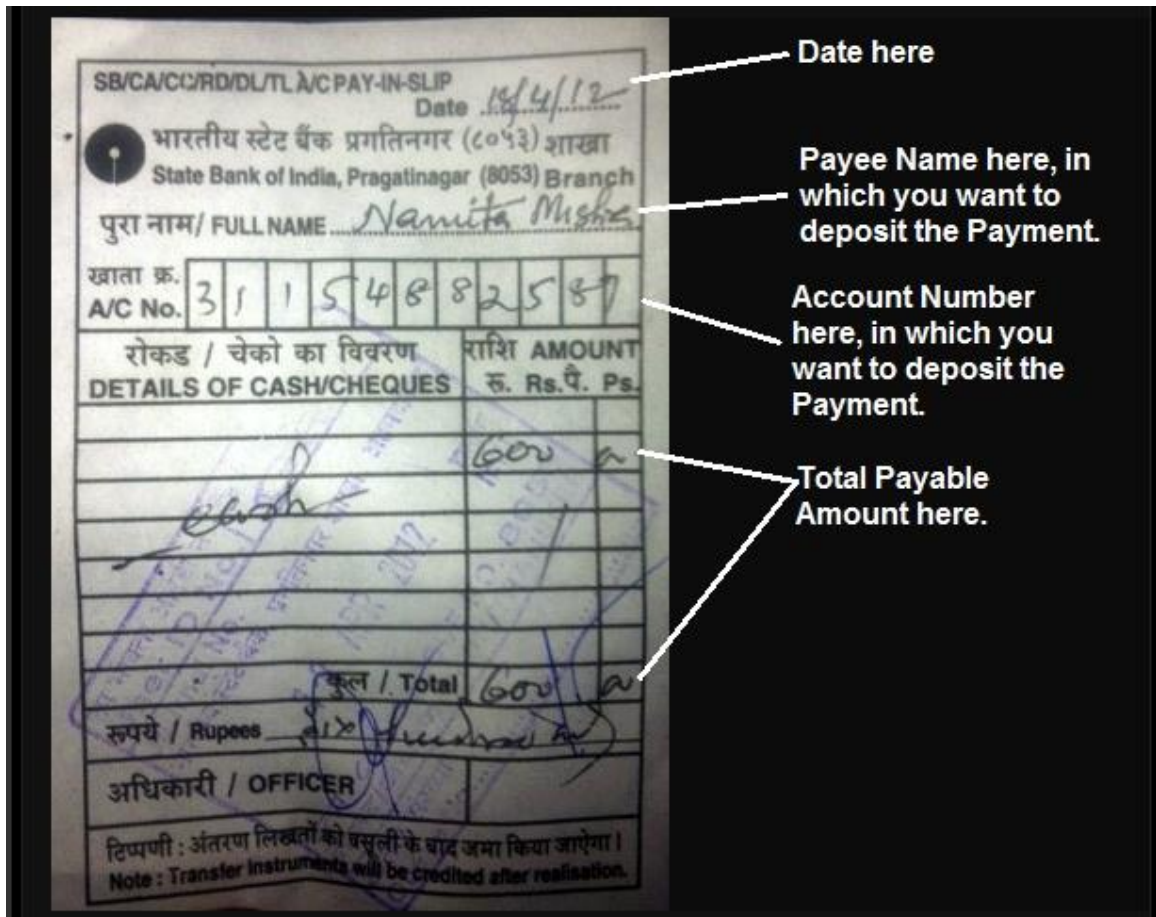
PNB Bank A/c no. : 4445000100034960
 Account Name : Kuldeep Chand
 Branch Name : Falna
 Address : College Road, Falna Station – 306116
 IFSC Code : PUNB0444500

जब आप Net-Banking के माध्यम से Payment करना चाहते हैं, तो आपको लगभग **8 से 24 घण्टे पहले** हमारे उस Account को **Beneficiary** के रूप में अपने Bank A/c से Link करना पड़ता है, जिसमें आप Payment Deposit करना चाहते हैं।

यदि आपके पास Net-Banking या Mobile-Banking की सुविधा नहीं है, तो आप हमारे किसी भी Bank A/c में **Total Payable Amount, Direct Deposit** भी कर सकते हैं।

जब आप **Direct Deposit** करना चाहते हैं, तब आपको आपके किसी भी नजदीकी Bank Branch में जाकर एक **Payment Deposit Slip Fill-Up** करना होता है, जिसमें आपको हमारे किसी भी Bank A/c की Information को Fill करना होता है, **जबकि Payment Deposit करवाने के लिए उसी Bank में आपका स्वयं का Account होना जरूरी नहीं है।**

उदाहरण के लिए यदि आप हमारे SBI Bank A/c में अपनी Selected पुस्तकों का **Total Payable Amount** Pay करने के लिए Bank में जाकर Direct Deposit करना चाहते हैं, तो आप जो **Payment Deposit Slip** Fill-Up करेंगे, वह निम्न चित्रानुसार करना होता है:



SB/CA/CC/RD/DL/TL A/C PAY-IN-SLIP
 Date 18/4/12

भारतीय स्टेट बैंक प्रगतिनगर (८०५३) शाखा
 State Bank of India, Pragatinagar (8053) Branch

पुरा नाम/ FULL NAME Nanita Mishra

खाता क्र. A/C No. 31154882587

रोकड / चेको का विवरण DETAILS OF CASH/CHEQUES	राशि AMOUNT रु. Rs. प. Ps.
600	
कुल / Total	600

रुपये / Rupees 600

अधिकारी / OFFICER

टिप्पणी : अंतरण लिखतों को वसूली के बाद जमा किया जाएगा।
 Note : Transfer Instruments will be credited after realisation.

इस चित्र द्वारा आप समझ सकते हैं कि Payment, Direct Deposit करने के लिए आपको हमारे किसी Bank A/c की Information को *Payment Deposit Slip* में Specify करना होता है, इसलिए उस Bank में आपका स्वयं का Bank A/c होना जरूरी नहीं होता।

Net-Banking, Mobile-Banking व **Direct Deposit** के अलावा किसी अन्य माध्यम से भी आप Payment कर सकते हैं। उदाहरण के लिए कुछ Banks अपनी ATM Machine द्वारा Direct Payment Transfer करने की सुविधा Provide करते हैं।

यदि आपके Bank का ATM Machine इस तरह से Payment Transfer करने की सुविधा देता है, तो आपको Bank में जाकर Payment Deposit Slip के माध्यम से Payment करने की जरूरत नहीं होती, बल्कि आप Bank के ATM Machine से भी Directly हमारे किसी भी Bank A/c में **Total Payable Amount** Transfer कर सकते हैं। इसी तरह से यदि आप चाहें, तो हमारे किसी भी Bank A/c में Check द्वारा भी Amount Direct Deposit कर सकते हैं।

यानी आप किसी भी तरीके से हमारे किसी भी Bank A/c में *Total Payable Amount* Deposit कर सकते हैं। लेकिन हम **Money-Order, Demand-Draft** या **Check** जैसे Manual माध्यमों से Payment Accept नहीं करते, क्योंकि इस तरह का Payment Clear होने में बहुत समय लगता है। जबकि Direct Deposit या Mobile अथवा Net-Banking के माध्यम से तुरन्त Payment Transfer हो जाता है, जिससे हम आपको आपकी Purchased EBooks **20** से **30 Minute** के दरम्यान आपके Order में Specified **Email** पर Send कर देते हैं।

Confirm the Payment

जब आप अपनी Selected पुस्तकों को खरीदने के लिए उपरोक्तानुसार किसी भी तरीके से “*Total Payable Amount*” हमारे किसी भी Bank A/c में Deposit कर देते हैं, तो Payment Deposit करते ही आपको हमें उसी Mobile Number से एक **Call/Miss Call/SMS** करना होता है, जिसे आपने Order Place करते समय “**Order Form**” में Specify किया था।

इसी Mobile Number के माध्यम से हमें पता चलता है कि आपने किन पुस्तकों के लिए Order किया है और उनका *Total Payable Amount* कितना है। साथ ही हमें ये भी पता चल जाता है कि आप द्वारा Purchase की जा रही पुस्तकें किस Email Address पर Send करनी है।

आपके *Total Payable Amount* को हम Net-Banking के माध्यम से अपने Bank A/c में Check करते हैं और यदि आपका *Total Payable Amount* हमारे किसी भी Bank A/c में Deposit हुआ होता है, तो हम आपको **30 Minute** के दरम्यान आपकी Ordered EBooks आपके Email पर Send कर देते हैं, जिसे आप अगले दिन 12AM तक Download कर सकते हैं।

यदि अभी भी आपको कोई बात ठीक से समझ में न आ रही हो या किसी भी तरह का Confusion हो, तो आप **097994-55505** पर **Call/Miss Call/SMS** कर सकते हैं। यथा सम्भव तुरन्त आपको Callback किया जाएगा और आपकी समस्या या Confusion का Best Possible Solution करने की कोशिश की जाएगी।

उम्मीद है, इस पुस्तक के Sample Chapters का Demo भी आपको पसन्द आया होगा और हमें पूरा विश्वास है कि पूरी पुस्तक आपको और भी ज्यादा पसन्द आएगी।