

21CSS101J

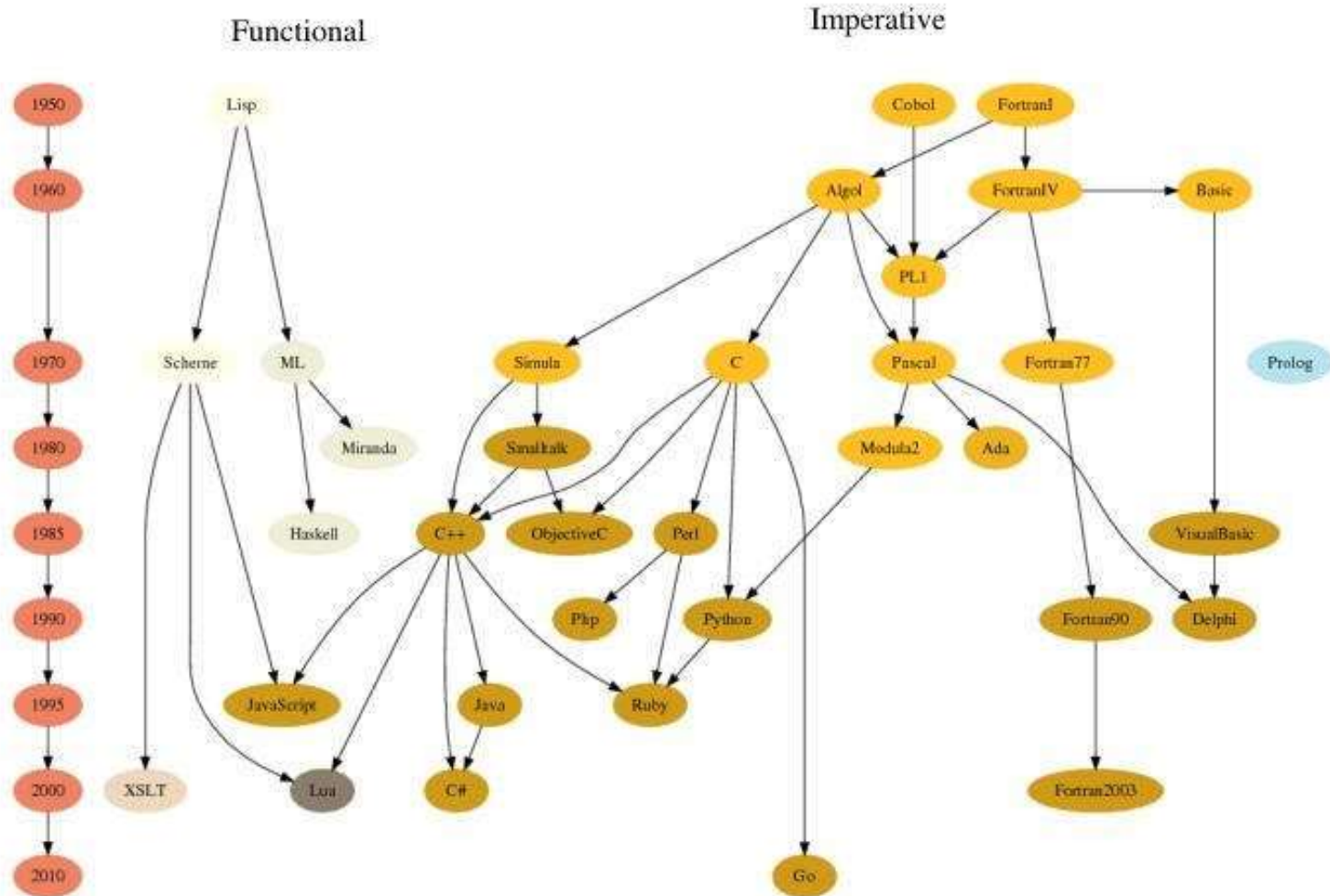
PROGRAMMING FOR PROBLEM SOLVING

UNIT-1

1. 1 Evolution of Programming & Languages

- ☐ A Computer needs to be given instructions in a programming language that it understands
- ☐ Programming Language
 - ☐ Artificial language that controls the behavior of computer
 - ☐ Defined through the use of syntactic and semantic rules
 - ☐ Used to facilitate communication about the task of organizing and manipulating information
 - ☐ Used to express algorithms precisely

Evolution of programming languages



1. 2 Problem Solving through Programming

- ❑ **Problem** - Defined as any question, something involving doubt, uncertainty, difficulty, situation whose solution is not immediately obvious
 - ❑ **Computer Problem Solving**
 - ❑ Understand and apply logic
 - ❑ Success in solving any problem is only possible after we have made the effort to understand the problem at hand
 - ❑ Extract from the problem statement a set of precisely defined tasks
-

1. 2 Problem Solving through Programming Contd...

i. Creative Thinking

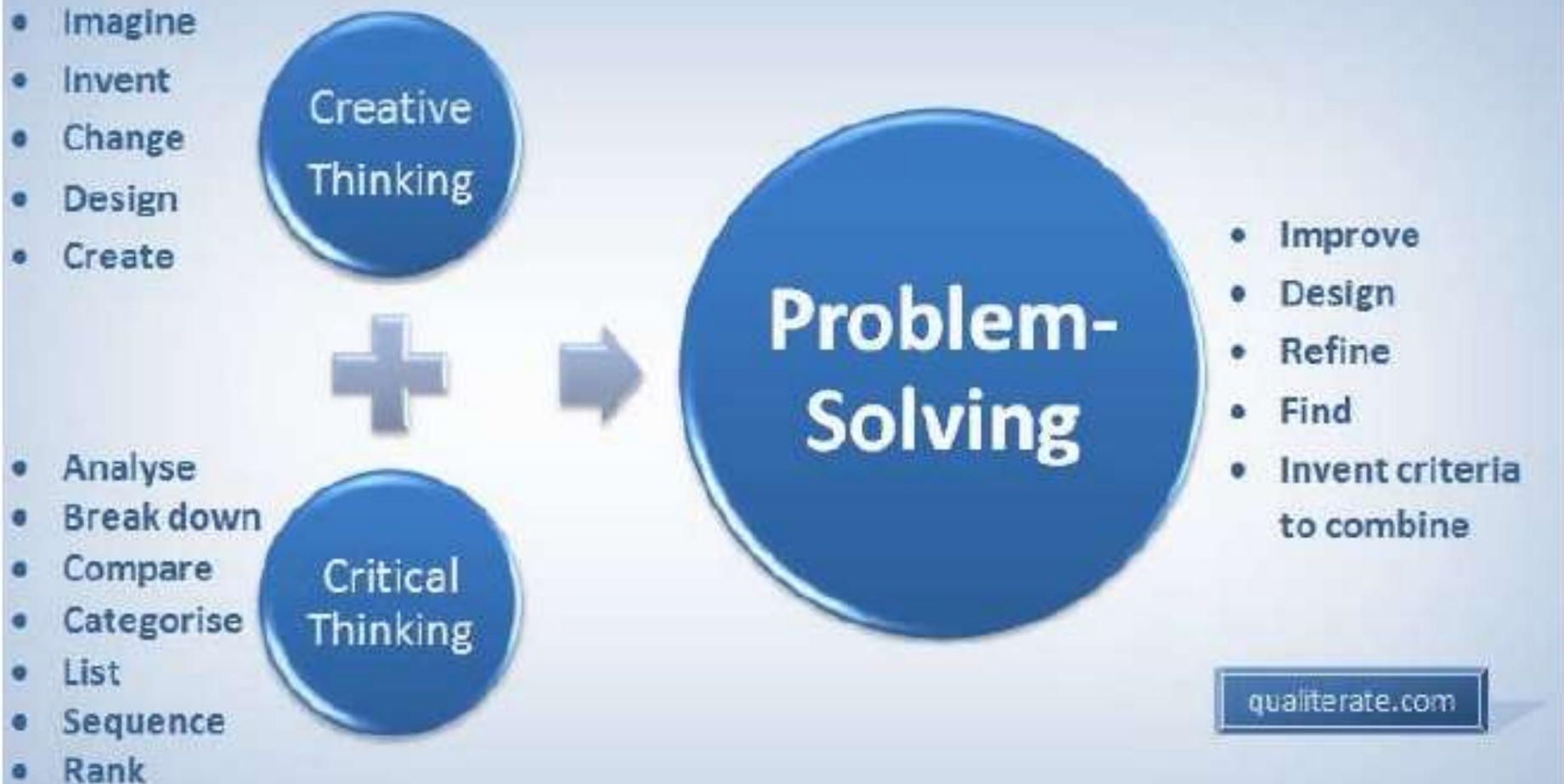
- ☐ Proven method for approaching a challenge or opportunity in an imaginative way
- ☐ Process for innovation that helps explore and reframe the problems faced, come up with new, innovative responses and solutions and then take action
- ☐ It is generative, nonjudgmental and expansive
- ☐ Thinking creatively, a lists of new ideas are generated

1. 2 Problem Solving through Programming Contd...

ii. Critical Thinking

- ☐ Engages a diverse range of intellectual skills and activities that are concerned with evaluating information, our assumptions and our thinking processes in a disciplined way so that we can think and assess information more comprehensively
- ☐ It is Analytical, Judgmental and Selective
- ☐ Thinking critically allows a programmer in making choices

1. 2 Problem Solving through Programming Contd...



1. 2 Problem Solving through Programming Contd...

- ❑ **Program** - Set of instructions that instructs the computer to do a task
- ❑ **Programming Process**
 - a) *Defining* the Problem
 - b) *Planning* the Solution
 - c) *Coding* the Program
 - d) *Testing* the Program
 - e) *Documenting* the Program

1. 2 Problem Solving through Programming Contd...



1. 2 Problem Solving through Programming Contd...

❑ A typical programming task can be divided into two phases:

i. Problem solving phase

❑ Produce an ordered sequence of steps that describe solution of problem this sequence of steps is called an **Algorithm**

ii. Implementation phase

❑ Implement the program in some programming language

❑ *Steps in Problem Solving*

a) Produce a general algorithm (one can use **pseudocode**)

1.2 Problem Solving through Programming Contd...

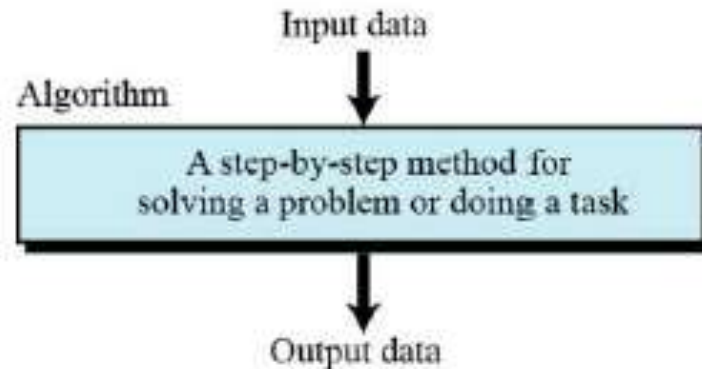
- b) Refine the algorithm successively to get step by step detailed *algorithm* that is very close to a computer language
- c) *Pseudocode* is an artificial and informal language that helps programmers develop algorithms
 - ❑ Pseudocode is very similar to everyday English

1.3 Creating Algorithms

- ❑ An informal definition of an algorithm is:



Algorithm: a step-by-step method for solving a problem or doing a task.



1.3 Creating Algorithms Contd...

- ❑ What are Algorithms for?
 - ❑ A way to communicate about your problem/solution with others
 - ❑ A possible way to solve a given problem
 - ❑ A "formalization" of a method, that will be proved
 - ❑ A mandatory first step before implementing a solution
- ❑ **Algorithm Definition** - "A finite sequence of unambiguous, executable steps or instructions, which, if followed would ultimately terminate and give the solution of the problem"

1.3 Creating Algorithms

☐ **Notations**

- ☐ Starting point
- ☐ Step Numbers – Positions in Algorithm
- ☐ Incoming Information – Input
- ☐ Control Flow – Order of evaluating Instructions
- ☐ Statements
- ☐ Outgoing Information – Output
- ☐ Ending Point

1. 3 Creating Algorithms Contd...

☐ *Properties of an algorithm*

- ☐ **Finite:** The algorithm must eventually terminate
- ☐ **Complete:** Always give a solution when one exists
- ☐ **Correct (sound):** Always give a correct solution

☐ *Rules of Writing an Algorithm*

- ☐ Be consistent
- ☐ Have well Defined input and output
- ☐ Do not use any syntax of any specific programming language

1. 3 Creating Algorithms Contd...

❑ Algorithm development process consists of five major steps

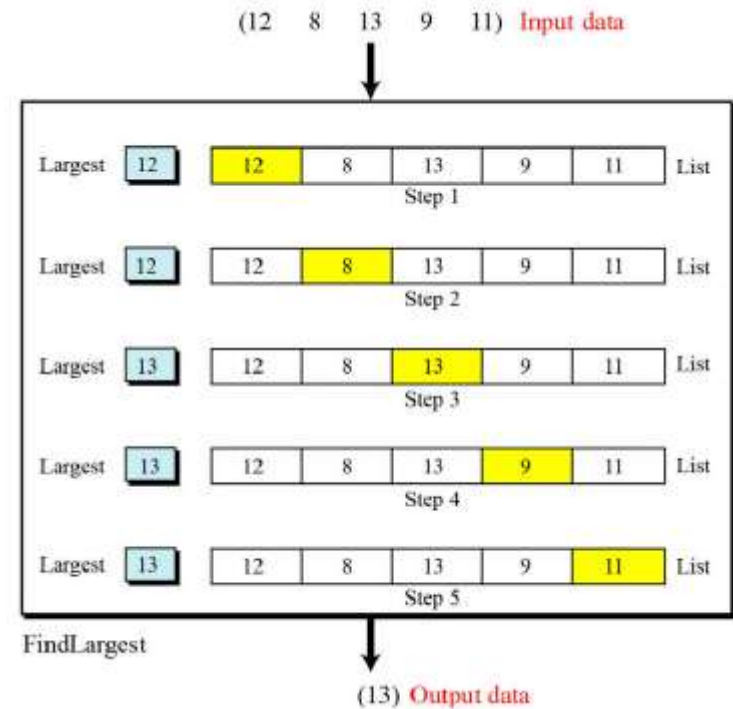
- ❑ **Step 1:** Obtain a description of the problem
- ❑ **Step 2:** Analyze the problem
- ❑ **Step 3:** Develop a high-level algorithm
- ❑ **Step 4:** Refine the algorithm by adding more detail
- ❑ **Step 5:** Review the algorithm

❑ **Problem**

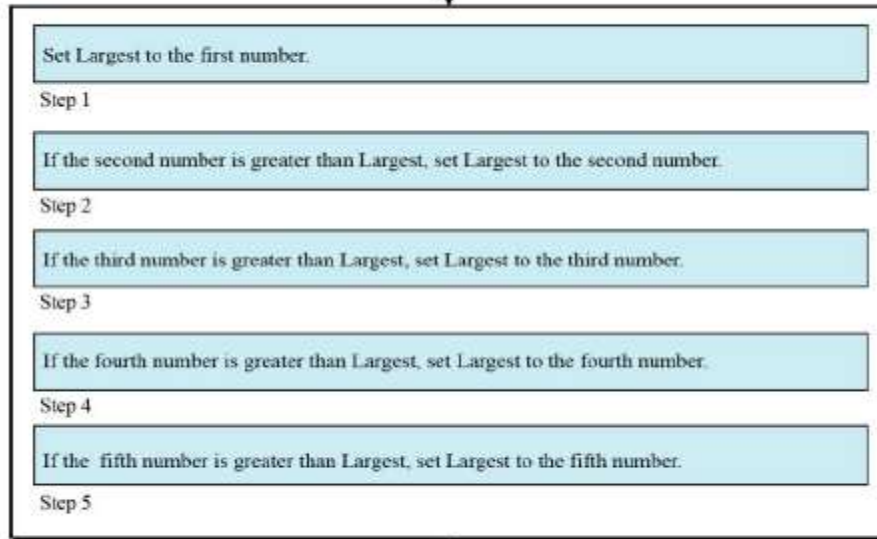
- a) Develop an algorithm for finding the largest integer among a list of positive integers
- b) The algorithm should find the largest integer among a list of any values
- c) The algorithm should be general and not depend on the number of integers

□ **Solution**

- To solve this problem, we need an intuitive approach
- First use a small number of integers (for example, five), then extend the solution to any number of integers
- The algorithm receives a list of five integers as input and gives the largest integer as output



(12 8 13 9 11) **Input data**

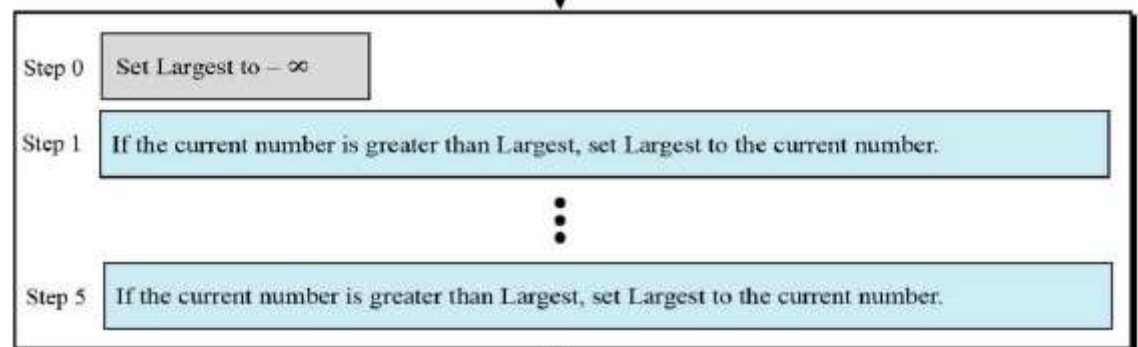


FindLargest



(13) **Output data**

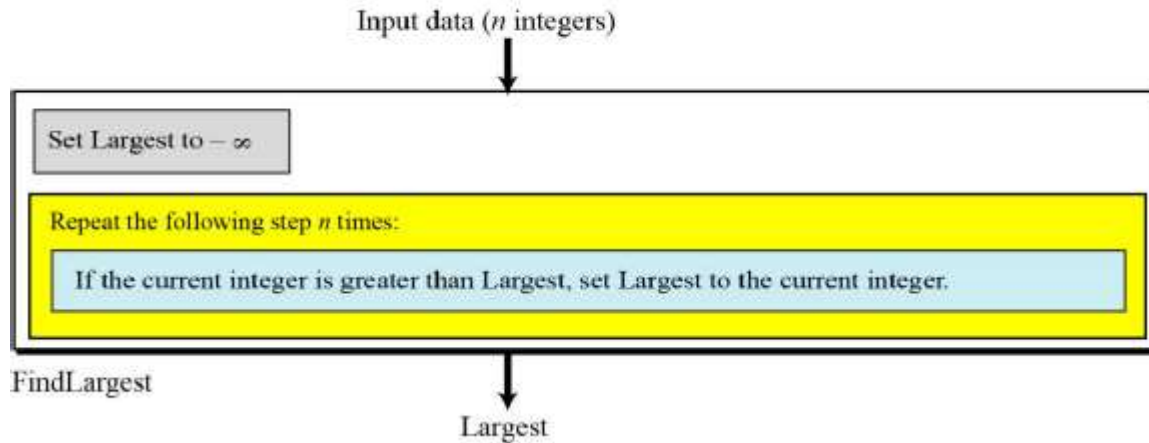
(12 8 13 9 11) **Input data**



FindLargest



(13) **Output data**



Example 2: Print 1 to 20

- ❑ **Step 1:** Start
- ❑ **Step 2:** Initialize X as 0,
- ❑ **Step 3:** Increment X by 1,
- ❑ **Step 4:** Print X,
- ❑ **Step 5:** If X is less than 20 then go back to step 2.
- ❑ **Step 6:** Stop

Example 3

Convert Temperature from Fahrenheit (°F) to Celsius (°C)

- ❑ **Step 1:** Start
- ❑ **Step 2:** Read temperature in Fahrenheit
- ❑ **Step 3:** Calculate temperature with formula $C = 5/9 * (F - 32)$
- ❑ **Step 4:** Print C
- ❑ **Step 5:** Stop

Example 4

Algorithm to Add Two Numbers Entered by User

- ❑ **Step 1:** Start
- ❑ **Step 2:** Declare variables num1, num2 and sum.
- ❑ **Step 3:** Read values num1 and num2.
- ❑ **Step 4:** Add num1 and num2 and assign the result to sum.
$$\text{sum} \leftarrow \text{num1} + \text{num2}$$
- ❑ **Step 5:** Display sum Step 6: Stop

□ Write an Algorithm to:

- 1) Find the Largest among three different numbers
- 2) Find the roots of a Quadratic Equation
- 3) Find the Factorial of a Number
- 4) Check whether a number entered is Prime or not
- 5) Find the Fibonacci Series

1. 4 Drawing Flowcharts

- ❑ Diagrammatic representation
- ❑ Illustrates sequence of operations to be performed
- ❑ Each step represented by a different symbol
 - ❑ Each Symbol contains short description of the Process
- ❑ Symbols linked together by arrows
- ❑ Easy to understand diagrams
- ❑ Clear Documentation
- ❑ Helps clarify the understanding of the process

Flowchart Symbols

Flowcharts are used to illustrate algorithms in order to aid in the visualisation of a program.

Flowcharts are to be read top to bottom and left to right in order to follow an algorithms logic from start to finish. Below is an outline of symbols used in flowcharts.



Terminator

Used to represent the Start and end of a program with the Keywords **BEGIN** and **END**.



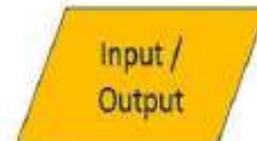
Decision

Used to split the flowchart sequence into multiple paths in order to represent **SELECTION** and **REPETITION**.



Process

An instruction that is to be carried out by the program.



Input / Output

Used to represent **data entry** by a user or the **display** of data by the program.



Arrow



























Indicates the flow of the algorithm pathways.



Subprogram

References another program within the program.

1. 4 Drawing Flowcharts Contd...

	Terminator Indicates the beginning or end of a program flow in your diagram.		Subroutine Indicates a predefined (named) process, such as a subroutine or a module.		Connector Indicates an inspection point.		Collate Indicates a step that organizes data into a standard format.
	Process Indicates any processing function.		Preparation Indicates a modification to a process, such as setting a switch or initializing a routine.		Off-page connector Use this shape to create a cross-reference and hyperlink from a process on one page to a process on another page.		Sort Indicates a step that organizes items list sequentially.
	Decision Indicates a decision point between two or more paths in a flowchart.		Display Indicates data that is displayed for people to read, such as data on a monitor or projector screen.		Off-page connector		Merge Indicates a step that combines multiple sets into one.
	Delay Indicates a delay in the process.		Manual input Indicates any operation that is performed manually (by a person).		Off-page connector		
	Data Can represents any type of data in a flowchart.		Manual loop Indicates a sequence of commands that will continue to repeat until stopped manually.		Off-page connector		
	Document Indicates data that can be read by people, such as printed output.		Loop limit Indicates the start of a loop. Flip the shape vertically to indicate the end of a loop.		Or Logical OR		Database Indicates a list of information with a standard structure that allows for searching and sorting.
	Multiple documents Indicates multiple documents.		Stored data Indicates any type of stored data.		Summing junction Logical AND		Internal storage Indicates an internal storage device.

1.4 Drawing Flowcharts Contd...

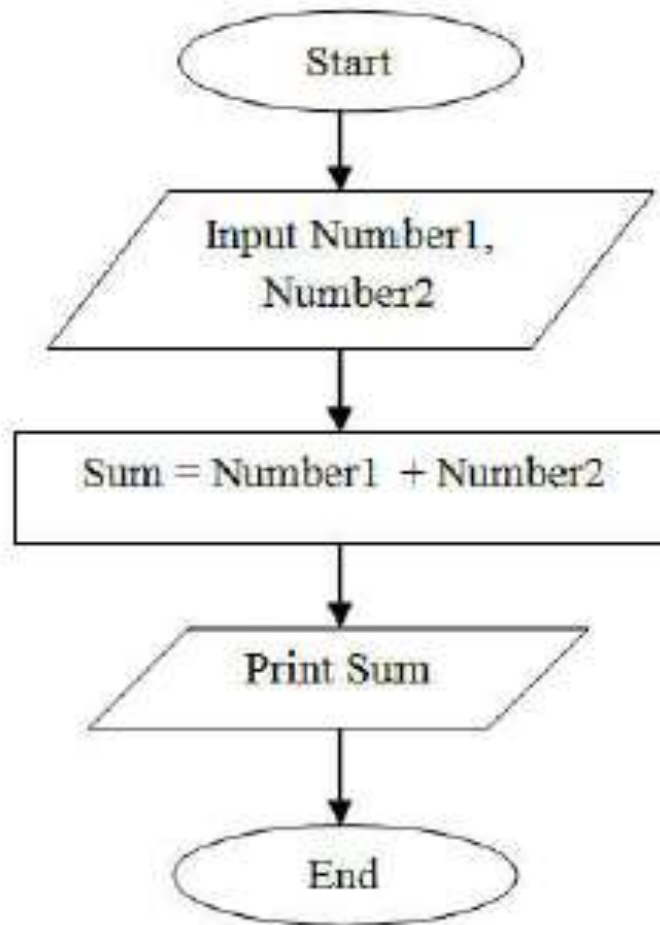
☐ **Guidelines for Preparing Flowchart**

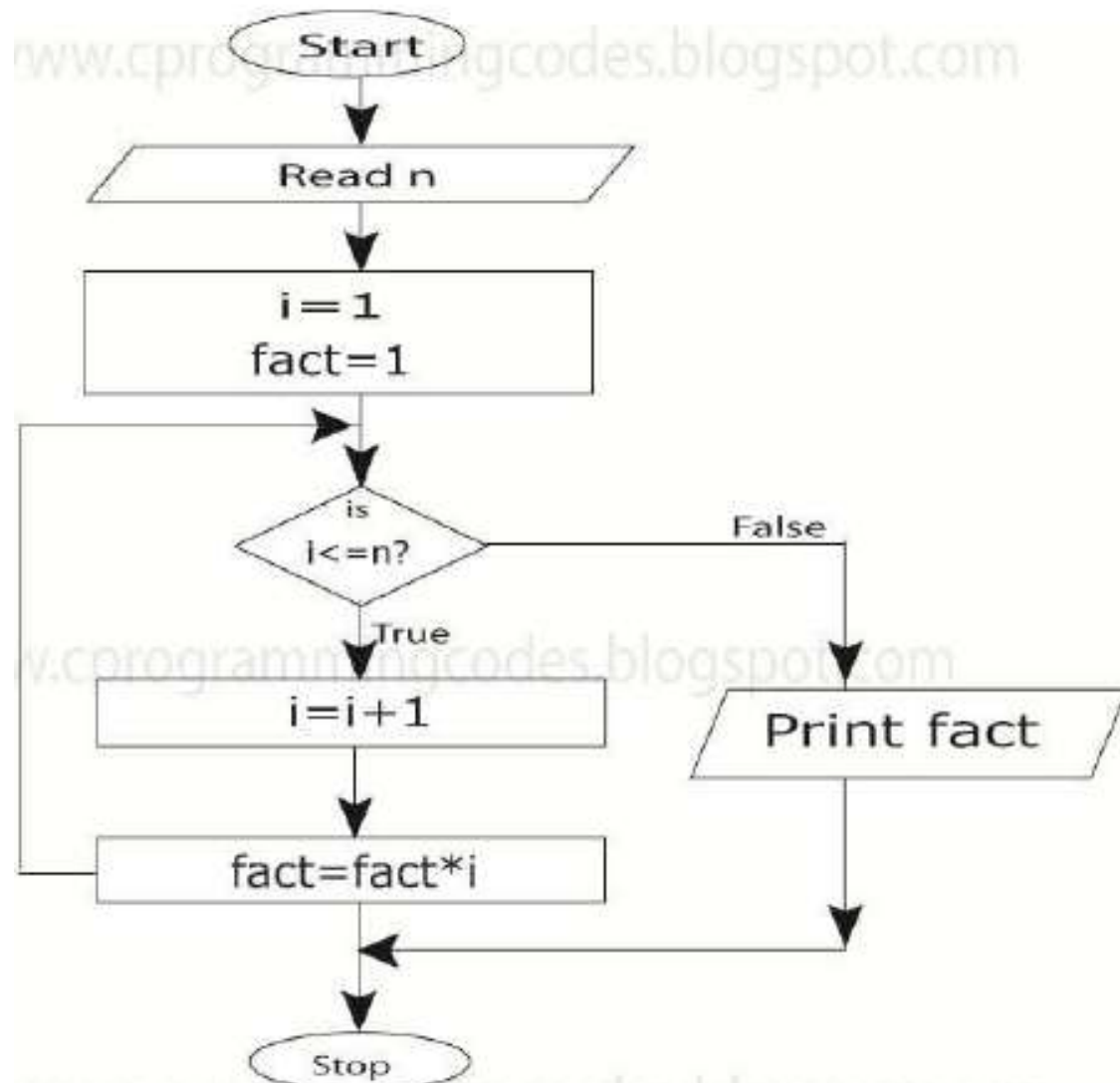
- ☐ Logical order of requirements
- ☐ Ensure that Flowchart has logical *Start* and *Stop*
- ☐ Direction is from Top to bottom
- ☐ Only one flow line is used with Terminal Symbol
- ☐ Only one flow line should come out of a Process symbol
- ☐ Only one flow line should enter a Decision symbol but multiple lines may leave the Decision symbol

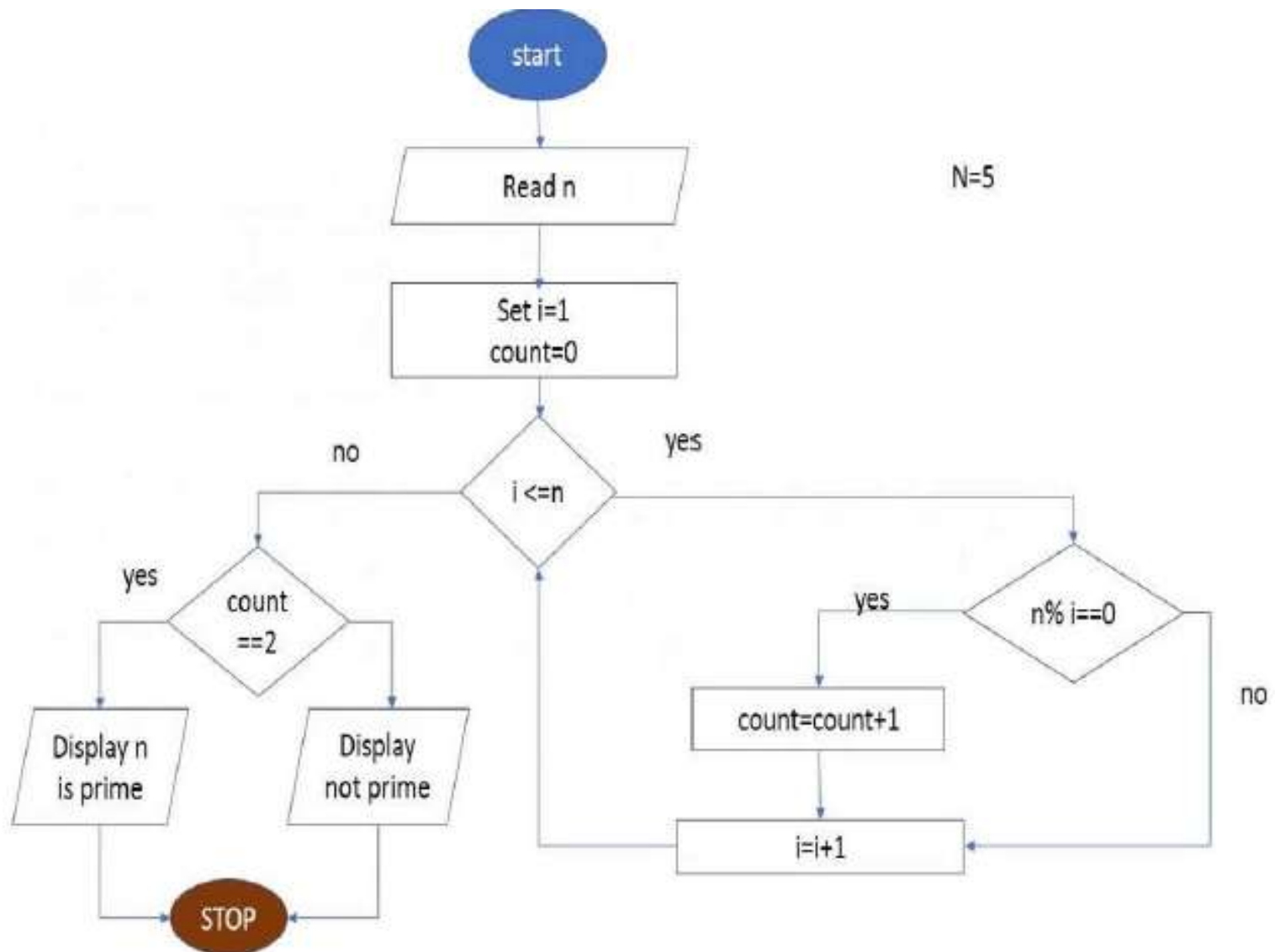
1. 4 Drawing Flowcharts Contd...

- ❑ **Guidelines for Preparing Flowchart Contd...**
 - ❑ Write briefly within Symbols
 - ❑ Use connectors to reduce number of flow lines
 - ❑ Avoid intersection of flow lines
 - ❑ Test Flowchart through simple test data
 - ❑ Clear, Neat and easy to follow

1. 4 Drawing Flowcharts Contd...







Pseudocode & History and Evolution of C

Writing Pseudo code

Pseudo – Imitation / False

Code - Instructions

Outline or a rough draft of the Program.

Simple English syntaxes.

Goal: To provide a high level description of the Algorithm

Benefit: Enables programmer to concentrate on Algorithm

- Similar to programming code .
- Explanation of the Algorithm .
- No specific Programming language symbolizations .
- Pseudo Code can be transformed into actual program code .

How to write a Pseudo-code?

Guidelines of Pseudo code

1. Sequence of operations need to be decided and arranged then should write the pseudo code accordingly.
2. Start with statement that establishes the aim of the Program.

Example

The program checks the given number is Odd or Even.

3. Indent the statement , as it helps to understand the execution mechanism of any decision control. The Pseudo code should ease the readability.

Example

if “yes”

Print response

“I am in”

if “No”

Print response

“I am not in”

4. Use proper Naming Conventions.
5. Use suitable sentence casings such
 - I. Camel Case –Methods.
 - II. Uppercase-Constants
 - III. Lowercase-Variables.
6. Don't be abstract with the Pseudo code . Make the Pseudo code elaborate and explanatory.
7. Use Structured programming such as i) if-then ii) for iii)while iv)case
8. Make sure the Pseudo code is very clear, understandable, finite and comprehend.
9. The pseducode should be understandable even for a layman or client. Don't make too much of technical complex terms.

Do's and Don'ts of Pseudo code

Do's

1. Use Control Structures.
2. Use appropriate naming conventions.
3. Indentation and white spaces are the key.
4. Keep it simple
5. Keep it concise.

Don'ts

1. Don't be too generalized.
2. Don't make the pseudo code abstract.

Advantages

- The Pseudo code can be easily converted into programming.
- Easy to understand for even non-programmers
- Syntax errors are not so important.
- Changes in the requirement can be easily incorporated in the pseudo code.
- Implements Structured concepts
- No special symbols
- No specific platform required, it can be typed in MS Word.

Disadvantage

- No standard is followed
- Compilation and Execution of the program cannot be done.
- It may be time consuming to produce result.

Examples of Pseudo code

1. Write a Pseudo code to add 2 numbers together and then display the result

Line1: Start Program

Line 2: Enter two numbers, A, B

Line 3: Add the numbers together

Line 4: Print Sum

Line 5: End Program

2. Write a Pseudo code to compute the area of a rectangle:

Line1 : Get the length, l, and width, w

Line2 : Compute the area = $l * w$

Line3: Display the area

Examples of Pseudo code Continues....

3. Compute the perimeter of a rectangle:

Line1 : Enter length, l

Line2 : Enter width, w

Line3 : Compute Perimeter = $2*l + 2*w$

Line4 : Display Perimeter of a rectangle

Examples of Pseudo code Continues....

4.Input examination marks and award grades

Use Variables: mark of type integer

If mark ≥ 80 DISPLAY “Distinction”

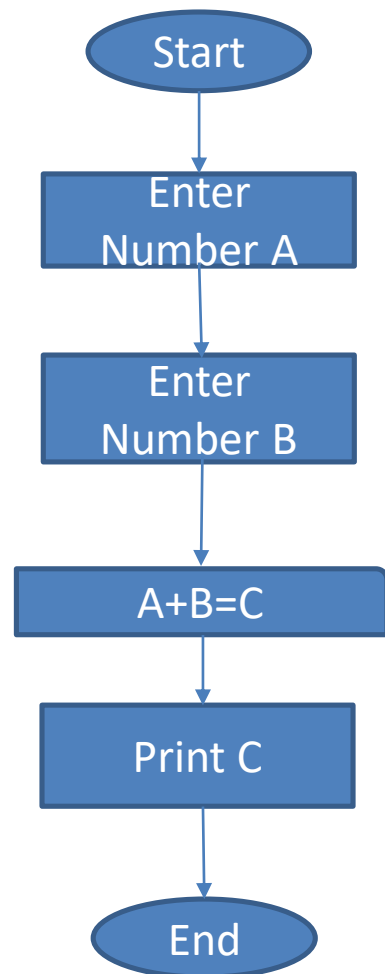
If mark ≥ 60 and mark < 80 DISPLAY “First Class”

If mark ≥ 50 and mark < 60 DISPLAY “Second Class”

If mark < 50 DISPLAY “Fail”

End Program

Flowchart to add 2 numbers and display the result



Line1: Start Program

Line 2: Enter two numbers, A, B

Line 3: Add the numbers together

Line 4: Print C

Line 5: End Program

History and Evolution of C

1. C – General Purpose Programming Language
2. Developed by Dennis Ritchie in 1972
3. Developed at Bell Laboratories
4. Developed to overcome the problems of previous languages such as B, BCPL, etc.
5. Structured Programming Language
6. C Program
 - ☐ Collection of Functions
 - ☐ Supported by C library



Father of C Programming : Dennis Ritchie

Born on	September 09,1941
Born In	Bronxville ,New York
Full name	Dennis MacAlistair Ritchie
Nick name	DMR
Nationality	American
Graduate From	Harvard University
Graduate In	Physics and Applied Mathematics
Dead On	October 12 2011

Evolution of C

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

Features of C Language

Features of C are

1. Simple
2. Machine Independent.
3. Mid-level programming language
4. Structured Programming language
5. Rich Library
6. Memory management
7. Speed
8. Pointer
9. Recursion
10. Extensible.

Advantages of C

1. Compiler based Language
2. Programming – Easy & Fast
3. Powerful and Efficient
4. Portable
5. Supports Graphics
6. Supports large number of Operators
7. Used to Implement Data structures

Disadvantages of C

1. Not a strongly typed Language
2. Use of Same operator for multiple purposes
3. Not Object Oriented

Input and Output Statements

standard input-output header file, named `stdio.h`

1. Input- To provide the data for the program
2. Output-To display the data on the screen
3. C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.
4. The Standard input-output file – Studio.h.
5. The input statement is called scanf()
6. The output statement is called printf().

Topic

Variables, Identifiers, Expressions, Single line comments and Multiline comments

- ⑩ Single line comment
- ⑩ Multi line comment
- ⑩ Variables
- ⑩ Identifiers
- ⑩ Expressions

USING COMMENTS

- ⑩ It is a good programming practice to place some comments in the code to help the reader understand the code clearly.
- ⑩ Comments are just a way of explaining what a program does. It is merely an internal program documentation.
- ⑩ The compiler ignores the comments when forming the object file. This means that the comments are non-executable statements.

Continued....

C supports two types of commenting:

- ⑩ // is used to comment a **single statement**. This is known as a line comment. A line comment can be placed anywhere on the line and it does not require to be specifically ended as the end of the line automatically ends the line.
- ⑩ /* is used to comment **multiple statements**. A /* is ended with */ and all statements that lie within these characters are commented.

EXAMPLE

```
// This is my first program in C
```

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

```
int main()
```

```
{
```

```
    printf("\n Welcome to the world of C ");
```

```
        /* the above statement prints the  
        statements given in double quotes*/
```

```
    return 0;
```

```
}
```

IDENTIFIERS

- ⑩ Identifiers are names given to program elements such as variables, arrays and functions.

Rules for forming identifier name:

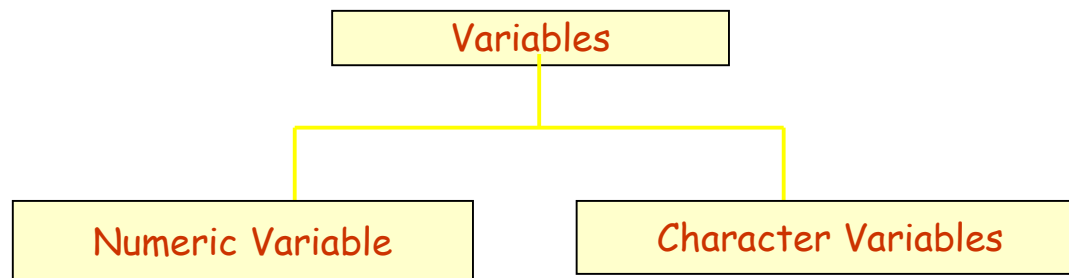
- it cannot include any special characters or punctuation marks (like #, \$, ^, ?, .., etc) except the underscore "_".
- There cannot be two successive underscores

KEYWORDS

- Keywords cannot be used as identifiers
- The names are case sensitive. So, example, “FIRST” is different from “first” and “First”.
- It must begin with an alphabet or an underscore.
- It can be of any reasonable length. Though it should not contain more than 31 characters.
- Example: roll_number, marks, name, emp_number, basic_pay, HRA, DA, dept_code

VARIABLES

- ⑩ Can provide one value for variables initialized in one statement:
 - ❑ `int x, y, z = 0;`
- ⑩ Each variable declared and then initialized with the value



VARIABLES

- ⑩ Numeric variables can be used to store either integer values or floating point values.
- ⑩ While an integer value is a whole numbers without a fraction part or decimal point, a floating point number, can have a decimal point in them.
- ⑩ Numeric values may also be associated with modifiers like short, long, signed and unsigned

VARIABLES

- ⑩ By default, C automatically a numeric variable signed..
- ⑩ Character variables can include any letter from the alphabet or from the ASCII chart and numbers 0 – 9 that are put between single quotes.

Example

```
int emp_num;  
float salary;  
char grade;  
double balance_amount;  
unsigned short int acc_no;
```

EXPRESSION

- linked to each other by the use of operators to compute a value. An operand can be a function reference, a variable, an array element or a constant.

Example:

- $a-b$;
- In the above expression, minus character (-) is an operator, and a, and b are the two operands.

- **There are four types of expressions exist in C:**
 - Arithmetic expressions Ex: $12/8 + 8 * 2$
 - Relational expressions Ex: $a != b$
 - Logical expressions Ex: $x > 10 \parallel y < 11$
 - Conditional expressions
Syntax: $\text{exp1} ? \text{exp2} : \text{exp3}$
- Each type of expression takes certain types of operands and uses a specific set of operators. Evaluation of a particular expression produces a specific value.

- 1. Keywords**
- 2. Constants**
- 3. Values, Names**
- 4. Scope**
- 5. Binding**

Keywords:

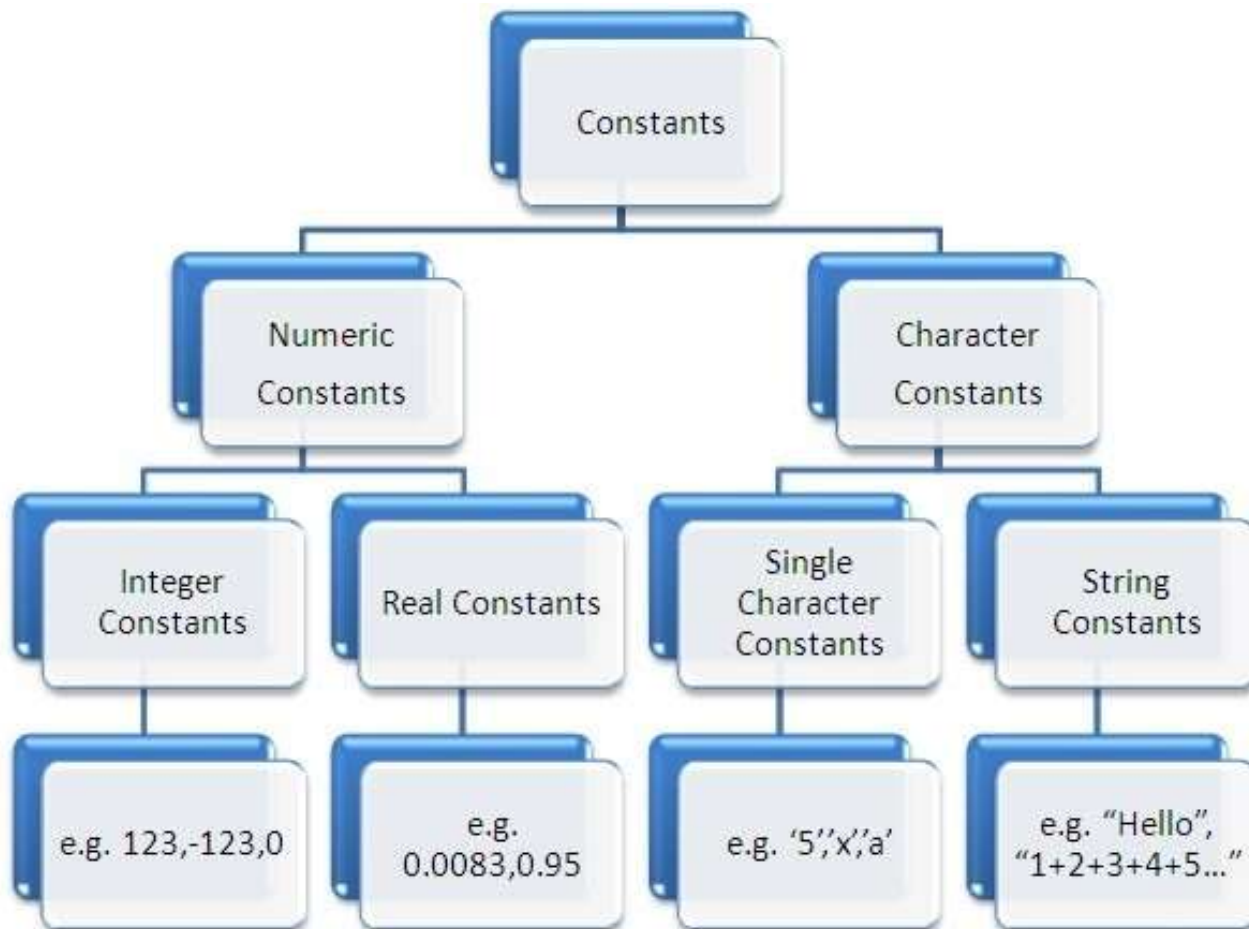
- ❑ **Keywords** – Conveys special meaning to Compiler
- ❑ Cannot be used as variable names
- ❑ The following are the 32 reserved C keywords:

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

Constants:

- ☐ Definition :Value does not change during execution
- ☐ Can be a Number (or) a Letter
- ☐ **Types**
 - ☐ Integer Constants
 - ☐ Real Constants
 - ☐ Character Constant
 - ☐ Single Character Constants
 - ☐ String Constants

Constants condit:



Values:

There are two kinds of expressions in C –

- lvalue – Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.
- rvalue – The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.
- Variables are lvalues and so they may appear on the left-hand side of an assignment.
- Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side. Take a look at the following valid and invalid statements –

`int g = 20; // valid statement`

`10 = 20; // invalid statement; would generate compile-time error`

Names:

Function names and variable names are both identifiers in C.

- The following are all the characters you can use to make a valid variable name:
 1. Characters A through Z and a through z.
 2. Digit characters 0 through 9, which can be used in any position except the first of a variable name.
 3. The underscore character (_).

For instance, stop_sign, Loop3, and _pause are all valid variable names.

Now, let's see what you cannot use in variable naming:

- A variable name cannot contain any C arithmetic signs.
- A variable name cannot contain any dots (.).
- A variable name cannot contain any apostrophes (`).
- A variable name cannot contain any other special symbols such as *, @, #, ?, and so on.

Some invalid variable names, for example, are, 4flags, sum-result, method*4, and what_size?.

Scope of Variable:

❑ *Definition*

- ❑ A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed
- ❑ **Variable Scope** is a region in a program where a variable is declared and used
- ❑ The ***scope*** of a variable is the range of program statements that can access that variable
- ❑ A variable is ***visible*** within its scope and ***invisible*** outside it

Scope of Variable:

- ❑ There are three places where variables can be declared
 - a) Inside a function or a block which is called **local** variables
 - b) Outside of all functions which is called **global** variables
 - c) In the definition of function parameters which are called **formal** parameters

Scope of Variable:

a) Local Variables

- ☐ Variables that are declared inside a function or block are called local variables
- ☐ They can be used only by statements that are inside that function or block of code
- ☐ Local variables are created when the control reaches the block or function containing the local variables and then they get destroyed after that
- ☐ Local variables are not known to functions outside their own

Scope of Variable:

/ Program for Demonstrating Local Variables */*

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

```
{
```

```
    /* local variable declaration */
```

```
    int a, b; int c;
```

```
    /* actual initialization */
```

```
    a = 10; b = 20;
```

```
    c = a + b;
```

```
    printf ("value of a = %d, b = %d and c = %d\n", a, b, c); return 0;
```

```
}
```

Scope of Variable:

b) Global Variables

- ☐ Defined outside a function, usually on top of the program
- ☐ Hold their values throughout the lifetime of the program
- ☐ Can be accessed inside any of the functions defined for the program
- ☐ Can be accessed by any function
 - ☐ That is, a global variable is available for use throughout the entire program after its declaration

Scope of Variable:

/ Program for Demonstrating Global Variables*/*

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

```
/* global variable declaration */
```

```
int g;
```

```
int main ( )
```

```
{
```

```
/* local variable declaration */
```

```
int a, b;
```

```
/* actual initialization */
```

```
a = 10; b = 20;
```

```
g = a + b;
```

```
printf ("value of a = %d, b = %d and g = %d\n", a, b, g); return 0;
```

```
}
```

Binding:

- ❑ A Binding is an association between an entity and an attribute
 - ❑ Between a variable and its type or value
 - ❑ Between a function and its code
- ❑ Binding time is the point at which a binding takes place
 - ❑ Types of Binding
 - a) Design Time
 - b) Compile Time
 - c) Link Time
 - d) Run Time

Binding:

a) Design Time

- ☐ Binding decisions are made when a language is designed
- ☐ Example
 - ☐ Binding of + to addition in C

b) Compile Time

- ☐ Bindings done while the program is compiled
- ☐ Binding variables to datatypes
- ☐ Example
 - ☐ `int a; float b; char c;`

Binding:

c) Link Time

- ☐ Compiled code is combined into a full program for C
- ☐ Example
 - ☐ Global and Static variables are bound to addresses

d) Run Time

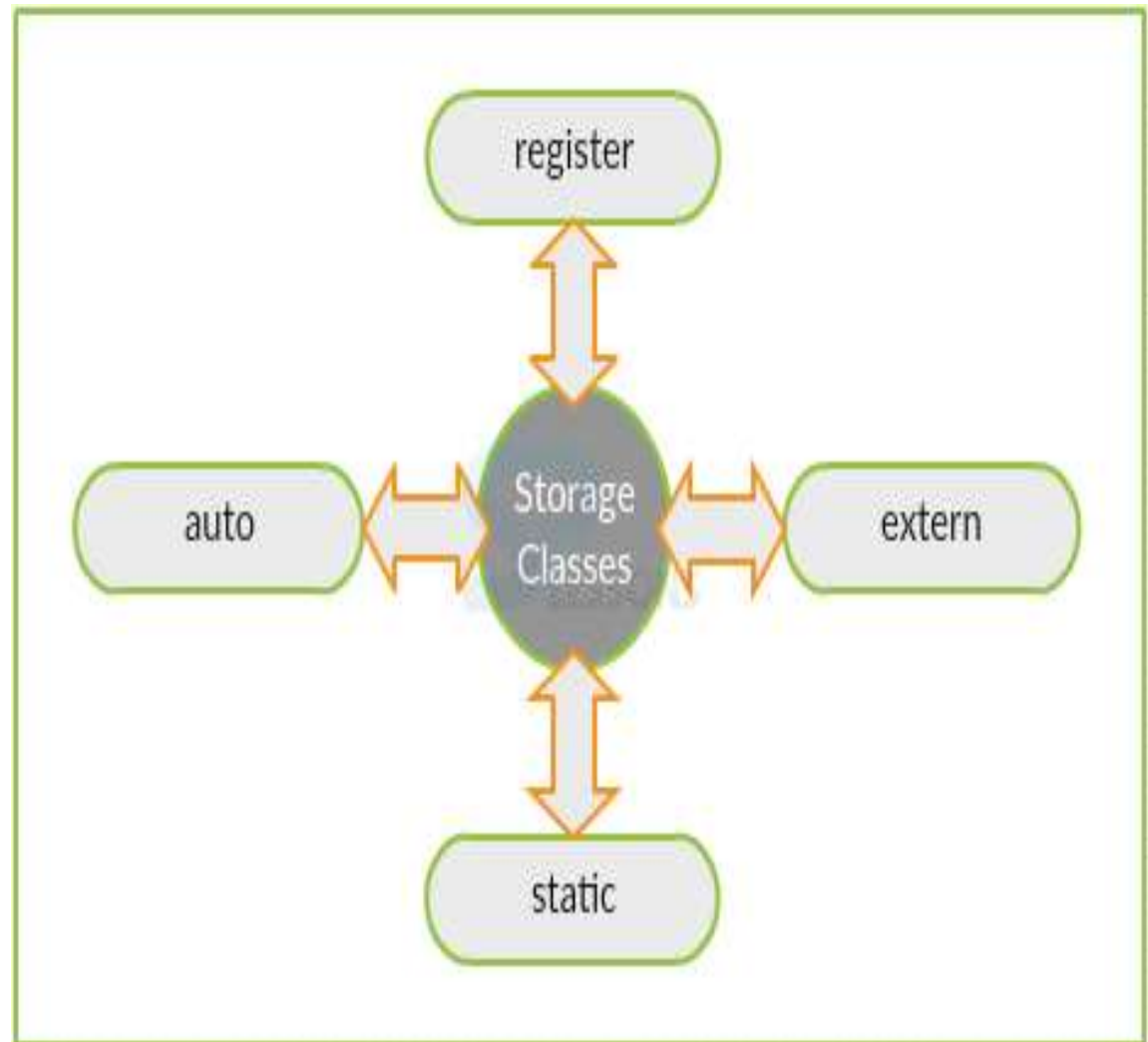
- ☐ Any binding that happens at run time is called *Dynamic*
- ☐ Any binding that happens before run time is called *Static*
- ☐ Values that are dynamically bound can change

Storage Classes



✓ Storage Class

- Where to store
- Default Initial Value
- Scope
- Life



Four Types of Storage Class

Auto

Every time new value

Static

Retains value between calls

Extern

Variable is defined Outside

Register

Value is Stored in CPU Register

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

Auto-Example

```
{  
int month;  
auto int month;  
}
```

.

Static-Example

```
#include <stdio.h>
/* function declaration */
void func(void);
    static int count = 5; /* global variable */
main() {
    while(count-->0)
    {
        func();
    }
    return 0;
}
/* function definition */
void func( void )
{
    static int i = 5; /* local static variable */
    i++;
    printf("i is %d and count is %d\n", i, count);
}
```

Output:

i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0

Extern-Example

First File: main.c

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

```
int count ;
```

```
extern void write_extern();
```

```
main() {
```

```
    count = 5;
```

```
    write_extern();
```

```
}
```

Second File: support.c

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

```
extern int count;
```

```
void write_extern(void) {
```

```
    printf("count is %d\n", count);
```

```
}
```

Output:

count is 5

Register-Example

```
{  
register int miles;  
}
```

Numeric Data Types : Integers and Floating Point

DATATYPES
IN C 

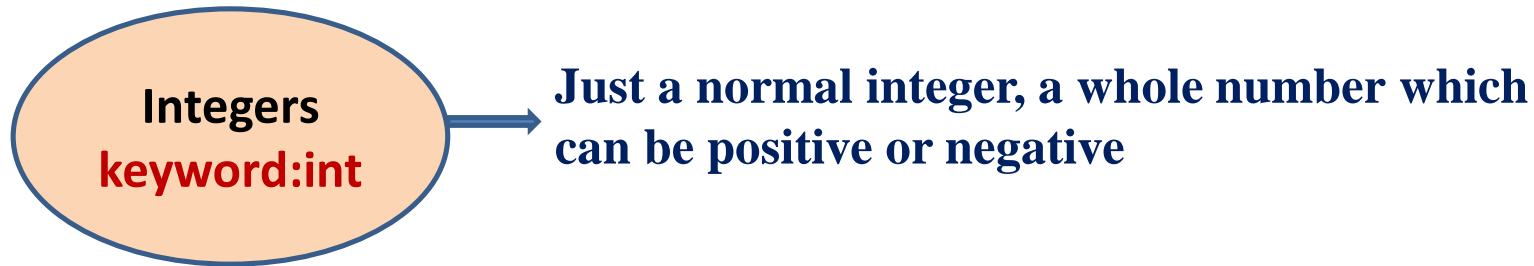
Data Types in C



Numeric Data Types : Integers and Floating Point



- Numeric Data means numbers.
- Numbers come in a variety of different types



- Explicit signs allowed
- Commas, decimal points and special signs not allowed
- Example:
 - 12
 - 45
 - 1274
 - 100000000
 - 3
 - 5735

Numeric Data Types : Integers and Floating Point



Floating point
keyword:float

It is used to store decimal numbers (numbers with floating point value) with single precision.

- Used for representing non-integer numbers
- fractional values
- “sign bit” for positive and negative numbers

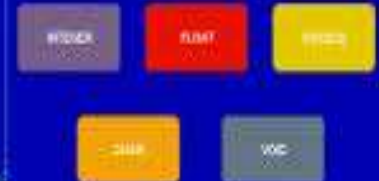
Examples:

```
float pi, _run_rate, Average_score;
```


Non Numeric Data Types : char and string

DATATYPES
IN C 

Data Types in C



Non Numeric Data Types: char and string

- **Non Numeric:**
 - Only Words are Counted
 - Cannot be manipulated mathematically using standard arithmetic operators.

Non Numeric	Character : 'a'
	String : "anything between double quotes"

CHARACTER VERSUS STRING

CHARACTER

A single letter, number, punctuation mark or symbol that can be represented using a computer

Character is a one element

Single quotes are used to represent a character

STRING

A one-dimensional array of characters terminated by a null character

String is a set of characters

Double quotes are used to represent a string

Numeric and Non-Numeric Data Types-Example

```
/* C program to Print Integer, Char, and Float value */
```

```
#include <stdio.h>
int main()
{
    int Integer;
    char Character;
    float InputFloat;

    printf(" Please Enter a Character : ");
    scanf("%c", &Character);

    printf(" Please Enter an Integer Value : ");
    scanf("%d", &Integer);

    printf(" Please Enter Float Value : ");
    scanf("%f", &InputFloat);

    printf(" \n The Integer Value that you Entered is : %d", Integer);
    printf(" \n The Character that you Entered is : %c", Character);
    printf(" \n The Float Value that you Entered is : %f", InputFloat);
    printf(" \n The Float Value with precision 2 is : %.2f", InputFloat);

    return 0;
}
```

```
Please Enter a Character : s
Please Enter an Integer Value : 125
Please Enter Float Value : 16.568975

The Integer Value that you Entered is : 125
The Character that you Entered is : s
The Float Value that you Entered is : 16.568975
The Float Value with precision 2 is : 16.57
```

@tutorialgateway.org

Operators



Topics

- Increment and decrement operators
- Assignment operators
- Comma ,Arrow
- Bitwise operators and SizeOf Operators

Operators

- Symbols which take one or more operands
- Perform arithmetic or logical computations.

Operators in C

	Operator	Type
Unary operator	++, --	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, , !	Logical operator
	&, , <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator	?:	Ternary or conditional operator

INCREMENT AND DECREMENT OPERATORS

- **Increment and decrement operators**
 - Unary operator
 - ++ (the increment operator)
 - -- (the decrement operator)
 - either a prefix or postfix position with different results

Increment/Decrement Operators		Let us assume X is a variable
Operator	Expression	Description
++	++X	Pre-Increment
	X++	Post-Increment
--	--X	Pre-decrement
	X--	Post-decrement

INCREMENT AND DECREMENT OPERATORS



Prefix and Postfix

++	Post-increment.	aNumber++
--	Post-decrement.	aNumber--
++	Pre-increment.	++yourNumber
--	Pre-decrement.	--yourNumber

Example:

```
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);
    return 0;
}
```

Output:

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

■ Assignment Operators

- Assign value to an variable and denoted by =
- Binary operator, operates on two operands
- Have Two Values – L-Value and R-Value.
- Also be used for logical operations

ASSIGNMENT OPERATORS



Operator	Example	Same as
=	$a = b$	$a = b$
+=	$a += b$	$a = a + b$
-=	$a -= b$	$a = a - b$
*=	$a *= b$	$a = a * b$
/=	$a /= b$	$a = a / b$
%=	$a \% = b$	$a = a \% b$

Comma (,) as Separator and Operator

- Separator -While declaration multiple variables and providing multiple arguments in a function
- Operator-assign multiple values to a variable

Example: As a Separator

```
int a,b,c;
```

(a, b, and c) are three different variables.

Example: As a Operator

```
a = 10,20,30;
```

```
b = (10,20,30);
```

value of a will be 10

value of b will be 30

Arrow Operator

- Accessing members of structure using pointer variable
- Access the structure members when we use pointer variable to access it

```
struct student  
{  
    char name[20],  
    int roll;  
}*ptr
```

Bitwise Operator

Bitwise Operator

- Used in bit level programming
- Manipulate bits of an integer expression
- Logical, shift and complement are three types

BITWISE OPERATORS

&	AND
	OR
^	Exclusive OR(XOR)
~	One's Complement (NOT)
>>	Shift right
<<	Shift left

Bit-wise Operators

Let us assume X and Y are two variables

Operator

Expression

Description

&

X & Y

To perform bit-wise AND operation

|

X | Y

To perform bit-wise OR operation

~

~ X

To perform bit-wise Not operation

^

X ^ Y

To perform bit-wise XOR operation

<<

X << Y

To perform bit-wise Left Shift

>>

X >> Y

To perform bit-wise Right Shift

Bitwise Operator



Example:

```
#include <stdio.h>
main()
{
    unsigned int a = 60;
    unsigned int b = 13;
    int c = 0;
    c = a & b;
    printf("Line 1 - Value of c is %d\n", c );
    c = a | b;
    printf("Line 2 - Value of c is %d\n", c );
    c = a ^ b;
    printf("Line 3 - Value of c is %d\n", c );
    c = ~a;
    printf("Line 4 - Value of c is %d\n", c );
    c = a << 2;
    printf("Line 5 - Value of c is %d\n", c );
    c = a >> 2;
    printf("Line 6 - Value of c is %d\n", c );
}
```

Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15

Sizeof Operator



- calculate the size of data type or variables.
- sizeof operator can be nested.
- sizeof operator will return the size in integer format.

```
#include <stdio.h>
int main()
{
    int integerVar;
    printf("Size of char = %d\n", sizeof(char));
    printf("Size of int = %d\n", sizeof(integerVar));
    printf("Size of expression (3+2.5) = %d\n",
    sizeof(3 + 2.5));
    return 0;
}
```

Output

Size of char = 1
Size of int = 4
Size of expression
(3+2.5) = 8

- ❖ **Relational and Logical Operators**
- ❖ **Conditional Operators**
- ❖ **Operator Precedence**
- ❖ **Pre / Post increment operators**

- ❖ The relational operators are used to compare two values.
- ❖ An expression which consists of a relational operator is known as a **relational expression**.
- ❖ The final result of a relational expression is either 0 or 1. ie, True or False.

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

❖ $==$ and $!=$ relational operators are called equality operators.

❖ General form

expression1 relational-operator expression2

❖ Example1: $x=1$, $y=2$ and $z=3$

Expression	Interpretation	Value
1. $x < y$ $1 < 2 \square \quad T$	true	1
2. $(x+y) \geq z$ $1+2 \geq 3 \square \quad T$	true	1
3. $(x+z) > (x+5)$ $1+3 > 1+5 \square \quad F$	false	0
4. $x \neq 1$	false	0
5. $z == 3$	true	1

❖ Example 2: $a=2$, $b=3.5$, and $c='z'$

Expression	Interpretation	Value
1. $a < 4$	true	1
2. $(a+b) \geq 10$	false	0
3. $c \neq 'p'$	true	1

❖ Example 3: $a=20$, $b=10$

if ($a > b$)

 printf("a is big");

else

 printf("b is big");

Output:

 a is big

- ❖ The logical operators are used to combine two or more conditions to make decisions.

Operators	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

- ❖ Example:

$x < y \ \&\& \ z \neq 5$

- ❖ An expression which combines two or more relational expressions is called as a **logical expression** or a **compound relational expression**.

- ❖ The common truth table of logical `&&`, `||` and `!` operators are given below: (where 1-true and 0-false)

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

- ❖ `&&`-AND returns 1 when all the conditions are true

A	B	A B
0	0	0
0	1	1
1	0	1
1	1	1

- ❖ `||`-OR returns 1 when one or more conditions are true

A	!A
0	1
1	0

- ❖ `!`-NOT 1 when condition is false

- Example: $x=1$, $y=2$, and $z=3$.

<u>Expression</u>	<u>Interpretation</u>	<u>Value</u>
1. $(x \leq 6) \&\& (z == 3)$ $(1 \leq 6) \&\& (3 == 3)$ $T \&\& T$	true	1
2. $(y != 6) \&\& (z >= (x + 4))$ $T \&\& F$	false	0
3. $(x \leq 3) (z >= 4)$ $T F$	true	1

Conditional Operators (Ternary operator)



- ❖ A ternary operator pair “? :” is available in C language.
- ❖ It can able to construct conditional expressions.
- ❖ The general form is

exp1 ? exp2 : exp3;

- ❖ Where exp1, exp2, and exp3 are expressions.
- ❖ First, exp1 is evaluated.
- ❖ If it is nonzero (true), then the expression exp2 is evaluated.
- ❖ Otherwise, exp1 will be false, exp3 is evaluated.

Conditional Operators (Ternary operator)



Example:

```
#include<stdio.h>
main()
{
    int a,b,c;
    printf("\nGive two numbers:");
    scanf("%d%d",&a,&b); /*a=15, b=10*/
    c = (a > b) ? a : b;
    printf("\nThe Biggest value: %d",c);
}
```

Output:

The Biggest value: 15

- ❖ An expression is evaluated from left to right.
- ❖ In C language, each operator has a “precedence” or priority associated with it.
- ❖ The operators at the higher level of priority are evaluated first.
- ❖ The operators of the same priority are evaluated either from left to right or from right to left, depending on the level. This is known as the **associativity property** of an operator.
- ❖ Example:

$$(3 * 5) + (4 / 2) - (1 * 2)$$

$$15 \quad + \quad 2 \quad - \quad 2$$

$$15$$

Operator Precedence

OPERATOR	DESCRIPTION	ASSOCIATIVITY	RANK
() []	Function call Array element reference	Left to right	1
+ - ++ -- ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference Address Size of an object Type cast		2
* / %	Multiplication Division Modulus	Left to right	3
+ -	Addition Subtraction	Left to right	4
<< >>	Left shift Right shift	Left to right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	6

- Increment and decrement operators are unary operators.
- They can add or subtract '1' from the operand.

Eg: $x = x + 1$ can be written as $++x$
result : $x=2$

- C languages has two types (pre and post) of each operator:
 1. Operator placed before variable (Pre)
 2. Operator placed after variable (Post)
- The increment/decrement operator is $++$ and $--$.

Increment and Decrement Operators Contd...

- Types:
 1. Pre Increment Operator
 2. Post Increment Operator
 3. Pre Decrement Operator
 4. Post Decrement Operator
- Syntax:
(pre)++var_name; (pre)--var_name;
(Or)
var_name++ (post); var_name -- (Post);
- Examples:
++a, --total, a--, i--

Increment and Decrement Operators Contd...

S. No	Operator type	Operator	Description
1	Pre Increment	<code>++x</code>	Value of x is incremented by 1 before assigning it to variable x.
2	Post Increment	<code>x++</code>	Value of x is incremented by 1 after assigning it to variable x.
3	Pre Decrement	<code>--x</code>	Value of x is decremented by 1 before assigning it to variable x.
4	Post Decrement	<code>x --</code>	Value of x is decremented by 1 after assigning it to variable x.

Increment and Decrement Operators Contd...

/ Program for Post Increment*/*

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x = 0;
    while (x++<3)
    {
        printf("%d,", x);
    }
    getch ( );
}
```

Output

1, 2, 3,

Increment and Decrement Operators Contd...

Program execution:

- ❖ **Step 1 :** In the program, value of x “0” is compared with 3 in while statement.
- ❖ **Step 2 :** Then, value of “x” is incremented from 0 to 1 using post- increment operator.
- ❖ **Step3:** Then, this incremented value “1” is assigned to the variable “x”.
- ❖ Above 3 steps are continued until while statement becomes ‘ false ‘ and output is displayed as “1, 2, 3”.

Increment and Decrement Operators Contd...

/ Program for Pre Increment*/*

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x = 0;
    while (++x<3)
    {
        printf("%d,", x);
    }
    getch ( );
}
```

Output

1, 2,

Increment and Decrement Operators Contd...

Program execution:

- ❖ **Step 1 :** In the program, value of “x” is incremented from 0 to 1 using pre-increment operator.
- ❖ **Step 2 :** Then, value of x “0” is compared with 3 in while statement.
- ❖ **Step3:** Then, this incremented value “1” is assigned to the variable “x”.
- ❖ Above 3 steps are continued until while statement becomes ‘ false ‘ and output is displayed as “1, 2”.

Conditional Operator

- C Programming Language provide '?' As Conditional operator
- These are called as 'Ternary operator'
- It has three operands.
- Syntax :
 Condition ? Expression1 : Expression2

- This equation is equal to :
if (condition)
 expression 1
else
 expression 2

- ## Working of Ternary Operator

The ternary operator is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison.

`<expression-1> ? <expression-2> : <expression-3> ;` Expression-1 \Rightarrow Conditional statement

Expression-2 \Rightarrow It will execute if condition in Expression-1 evaluate as TRUE.

Expression-3 \Rightarrow It will execute if Condition in Expression-1 evaluate as FALSE.

Example :

$\text{max} = (a > b) ? a : b ;$

This is equivalent to,

if $(a > b)$ $\text{max} = a;$

else $\text{max} = b;$

Note - Ternary operator is right associative.

Examples

```
#include <stdio.h>

int main()
{
    int x=1, y ;
    y = ( x ==1 ? 2 : 0 ) ;
    printf("x value is %d\n", x);
    printf("y value is %d", y);
}
```

Output :

x value is 1

y value is 2

Example

```
#include<stdio.h>
int main( )
{
int num;
printf("Enter the Number : ");
scanf("%d",&num);
(num%2==0)?printf("Even\n"):printf("Odd");
}
```

Output

Enter the Number : 10

Even

Example

```
#include<stdio.h>

int main( )
{
    int age;
    printf(" Please Enter your age here: \n ");
    scanf(" %d ", &age);
    (age >= 18) ? printf(" You are eligible to Vote ") : printf(" You are not eligible to
    Vote ");
    return 0;
}
```

Output

Please Enter your age here: 19

You are eligible to Vote

Examples

```
#include<stdio.h>
int main( )
{
int a, b, max;
printf("Enter a and b: ");
scanf("%d%d", &a, &b);
max = a > b ? a : b;

printf("Largest of the two numbers = %d\n", max);
return 0;
}
```

Output

Enter a and b: 10 20
Largest of the two numbers = 20

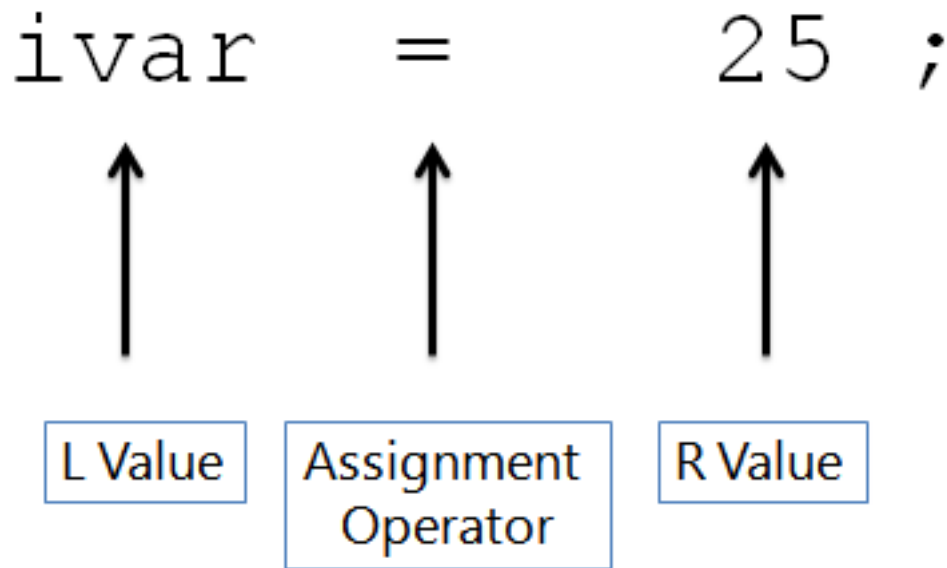
Assignment Operator

- Uses the assignment operator (=)
- General syntax: `variable_name = expression`
- Assignment Operator is binary operator which operates on two operands
- Assignment Operator have Two Values – L-Value and R-Value
 - Operator = copies R-Value into L-Value
- Left of = is called L-value, must be a modifiable variable
- Right of = is called R-value, can be any expression

Operator	Example	Equivalent Expression
=	$m = 10$	$m = 10$
+=	$m += 10$	$m = m + 10$
-=	$m -= 10$	$m = m - 10$
*=	$m *= 10$	$m = m * 10$
/=	$m /=$	$m = m/10$
% =	$m \% = 10$	$m = m \% 10$
<<=	$a <<= b$	$a = a << b$
>>=	$a >>= b$	$a = a >> b$
>>>=	$a >>>= b$	$a = a >>> b$
& =	$a \& = b$	$a = a \& b$
^ =	$a \wedge = b$	$a = a \wedge b$
=	$a = b$	$a = a b$

- L-Value stands for left value
- L-Value of Expressions refer to a memory locations
- In any assignment statement L-Value of Expression must be a container(i.e. must have ability to hold the data)
- Variable is the only container in C programming thus L Value must be any Variable.
- L Value cannot be a Constant, Function or any of the available data type in C.

Diagram Showing L-Value of Expression :



L-Value and R-Value of Expression Contd...

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

```
int main( )  
{
```

```
int num;
```

```
num=5;
```

```
return(0);
```

```
}
```

Example of L-Value Expression

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

```
int main( )  
{
```

```
int num;
```

```
5 = num; //Error
```

```
return(0);
```

```
}
```

L-value cannot be a Constant

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

```
{
```

```
const num;
```

```
num = 20; //Error
```

```
return(0);
```

```
}
```

L-value cannot be a Constant Variable

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

```
#define MAX 20
```

```
int main( )
```

```
{
```

```
MAX = 20; //Error
```

```
return(0);
```

```
}
```

L-value cannot be a MACRO

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

```
enum {JAN,FEB,MARCH};
```

```
int main( )
```

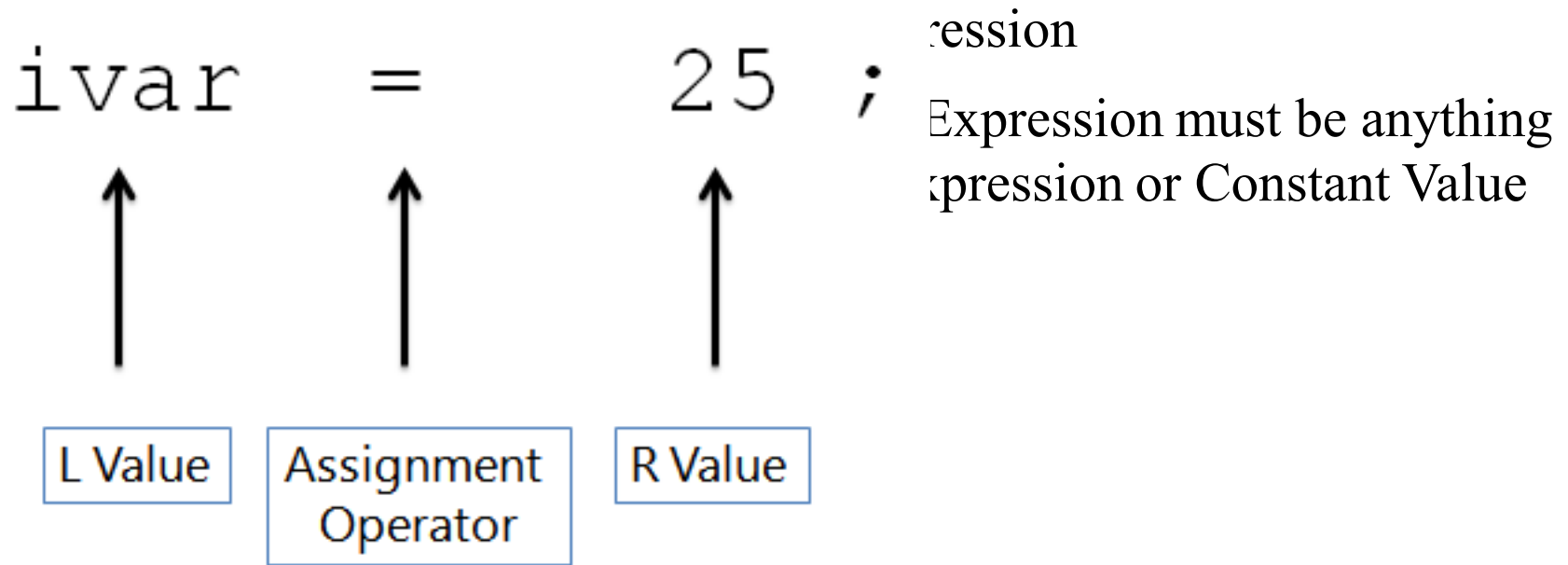
```
{
```

```
JAN = 20; //Error
```

```
return(0);
```

```
}
```

L-value cannot be a Enum Constant



Examples of R-Value of Expression	
Variable	Constant
Function	Macro
Enum Constant	Any other data type

R value may be a Constant or Constant Expression

R value may be a MACRO

R Value may be a variable

References

- Maureen Sprankle, “Problem Solving and Programming Concepts”, Pearson, 7th Edition, 2011
- E. Balagurusamy, “Programming in ANSI C”, Tata McGrawHill, 5th Edition, 2011.
- Y.P. Kanetkar, “Let us C”, BPB Publications, 8th Edition, 2008.
- Steve Oualline, “ Practical C Programming”, O’Reilly Publishers, 2011.
- Byron Gottfried, “Programming with C”, Schaum’s Outline Series, 2nd Edition, 2000.