SRM
INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

21CSS101J – Programming for Problem Solving

Unit I

SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

Prepared by:

DR.D.SHINY IRENE

ASSITANT PROFESSOR(CTECH)

SRMIST-KTR

SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

LEARNING RESOURCES	
S. No	**TEXT BOOKS**
1.	*Zed A Shaw, Learn C the Hard Way: Practical Exercises on the Computational Subjects You Keep Avoiding (Like C), Addison Wesley, 2015*
2.	*W. Kernighan, Dennis M. Ritchie, The C Programming Language, 2nd ed. Prentice Hall, 1996*
3.	*Bharat Kinariwala, Tep Dobry, Programming in C, eBook*
4.	*http://www.c4learn.com/learn-c-programming-language/*

SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

UNIT I (TOPICS COVERED)

Evolution of Programming & Languages - Problem Solving through Programming - Creating Algorithms - Writing Pseudocode - Evolution of C language, its usage history - Input and output functions: Printf and scanf - Variables and identifiers – Expressions - Single line and multiline comments - Constants, Keywords - Values, Names, Scope, Binding, Storage Classes - Numeric Data types: integer -

SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

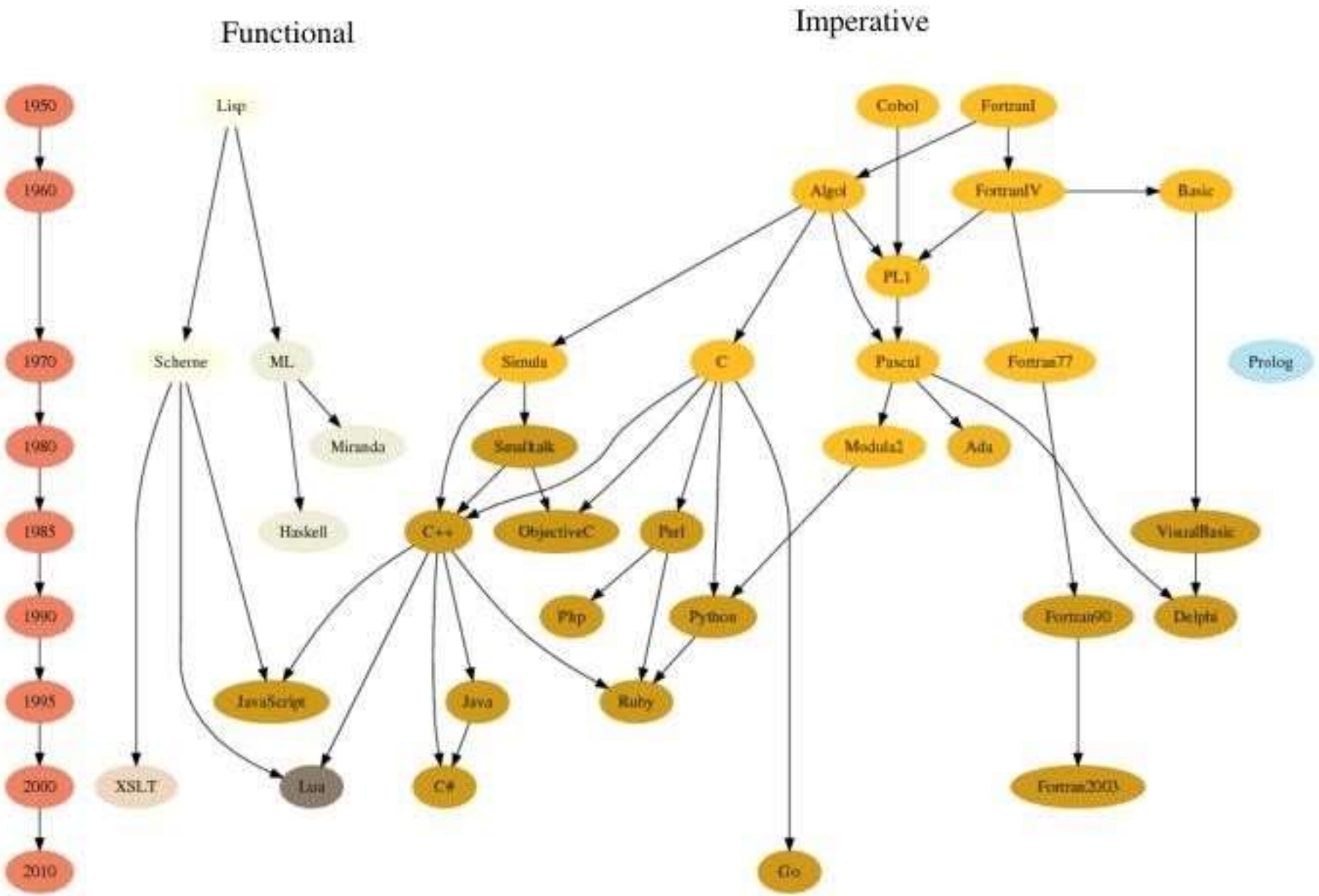
UNIT I (TOPICS COVERED)

floating point - Non-Numeric Data types: char and string - Increment and decrement operator - Comma, Arrow and Assignment operator - Bitwise and Sizeof operator Arithmetic, Relational and logical Operators, Condition Operator, Operator Precedence - Expressions with pre / post increment operator

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



Evolution of Programming & Languages Contd…

Period	Programming Langugaes
1950's	Creation of high-level languages
1960's	Forth. Simula I. Lisp, Cobol
1970's	Pascal, C language
1980's	ML. Smalltalk, C++
1990's	Java, Perl, Python languages
2000	Internet Programming
2010	Concurrency and asynchronicity. JavaScript and Go language

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

Problem Solving through Programming Contd…

i. Creative Thinking

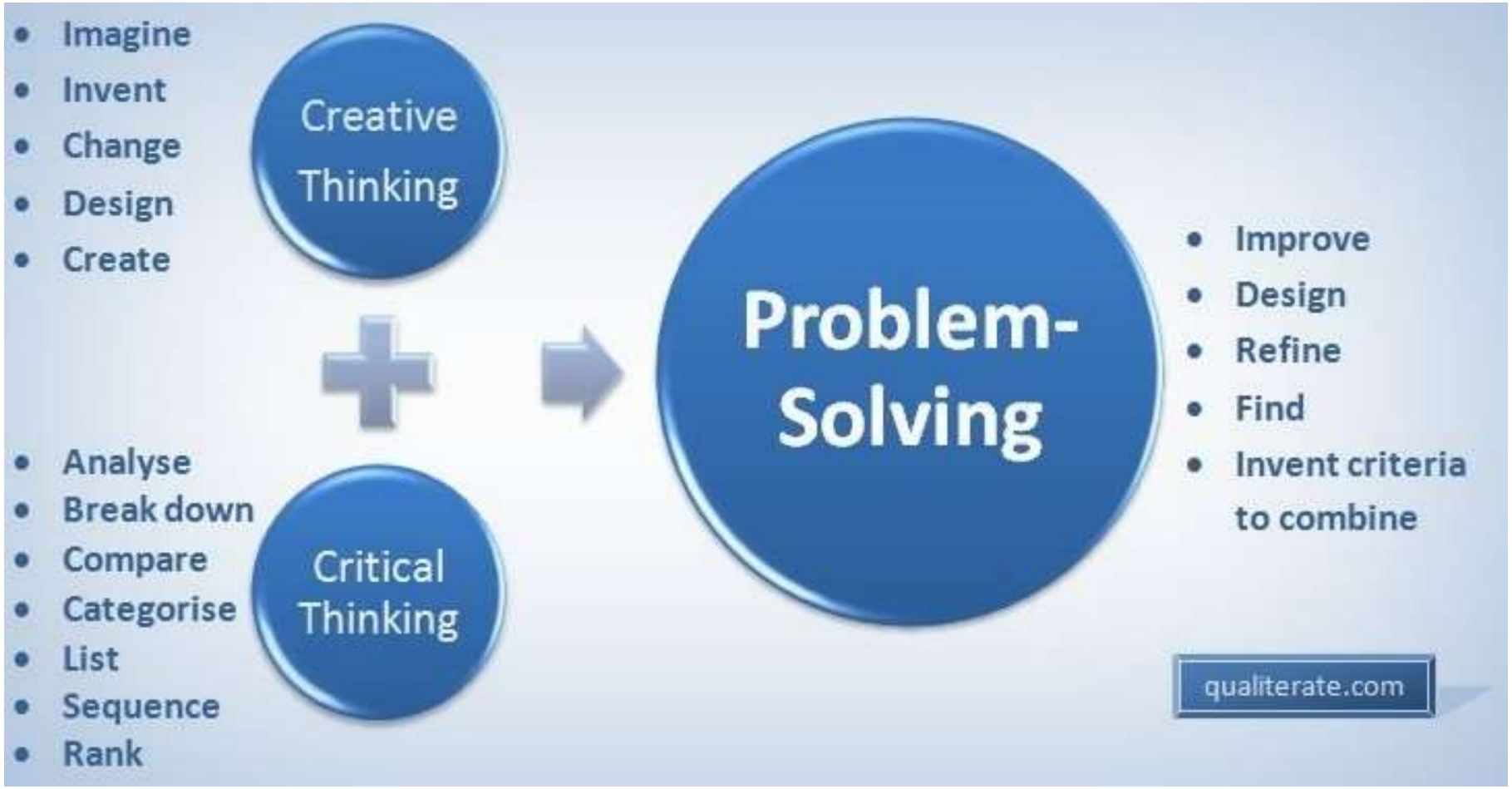
- Proven method for approaching a challenge or opportunity in an imaginative way
- Process for innovation that helps explore and reframe the problems faced, come up with new, innovative responses and solutions and then take action
- It is generative, nonjudgmental and expansive
- Thinking creatively,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

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

Problem Solving through Programming Contd…

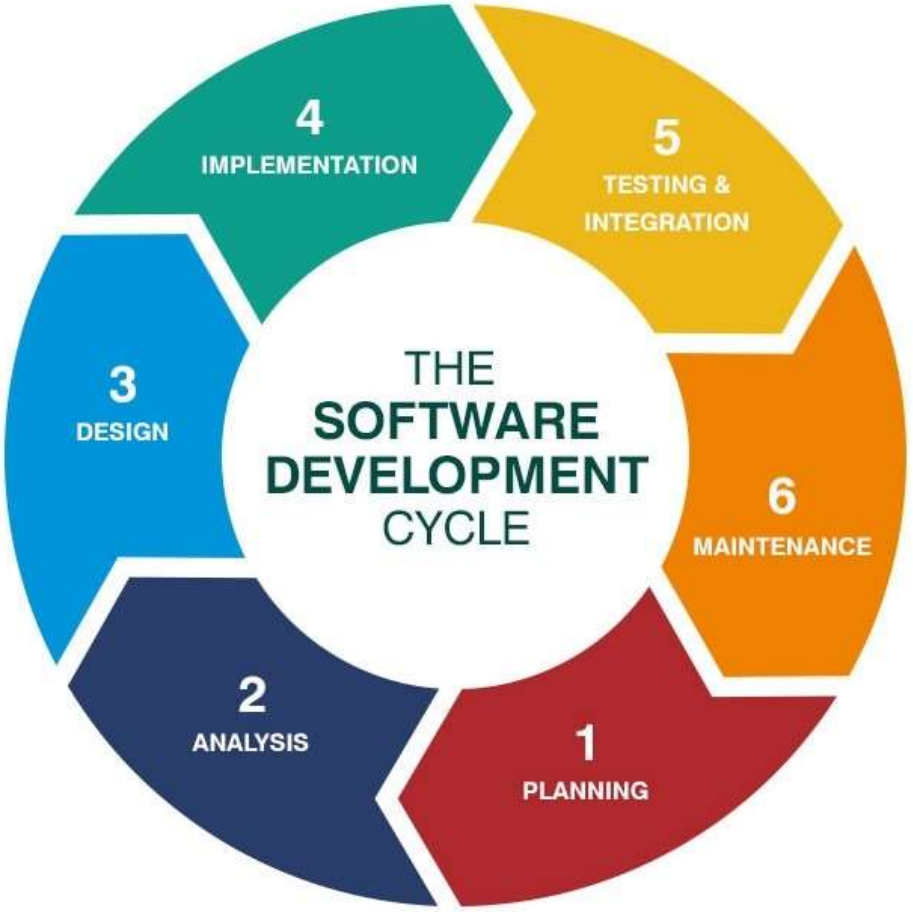


Problem Solving through Programming Contd…

- ***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

Problem Solving through Programming Contd…



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***)

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

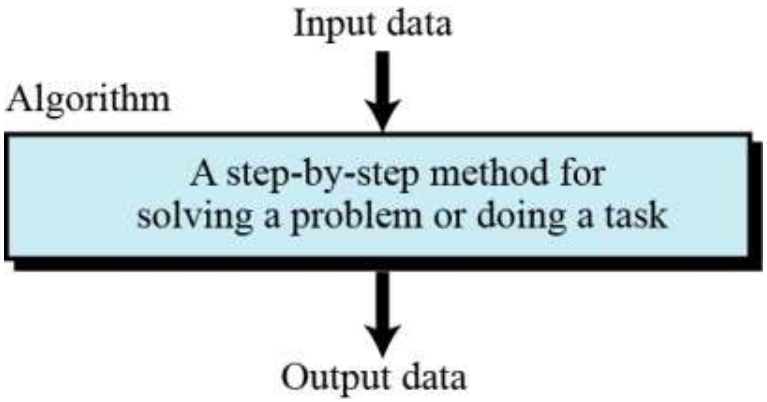
c) ***Pseudocode*** is an artificial and informal language that helps programmers develop algorithms

- Pseudocode is very similar to everyday English

Creating Algorithms

- An informal definition of an algorithm is:

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



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"

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

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

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

Creating Algorithms Contd…

Example

- ***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

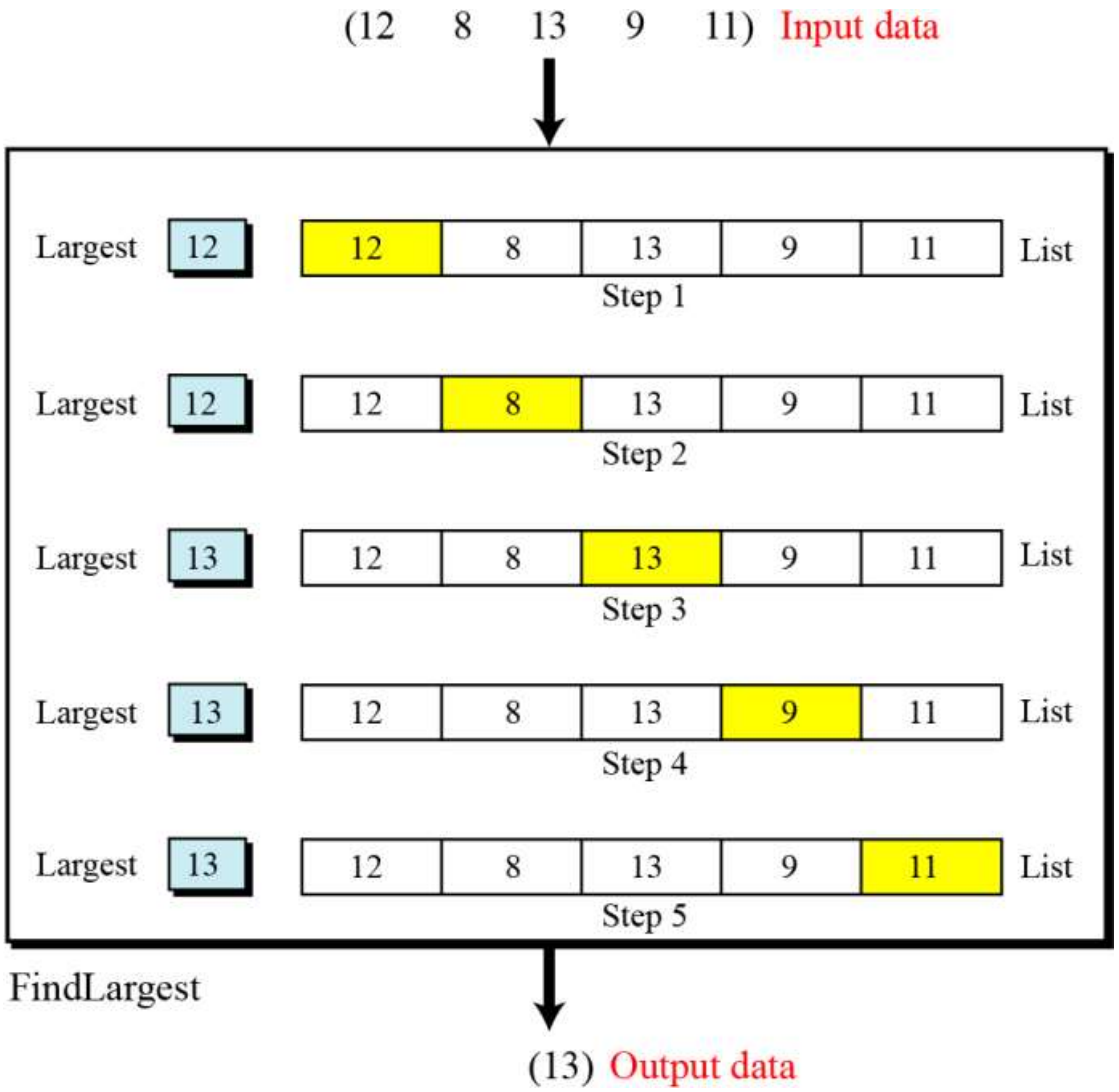
Creating Algorithms Contd…

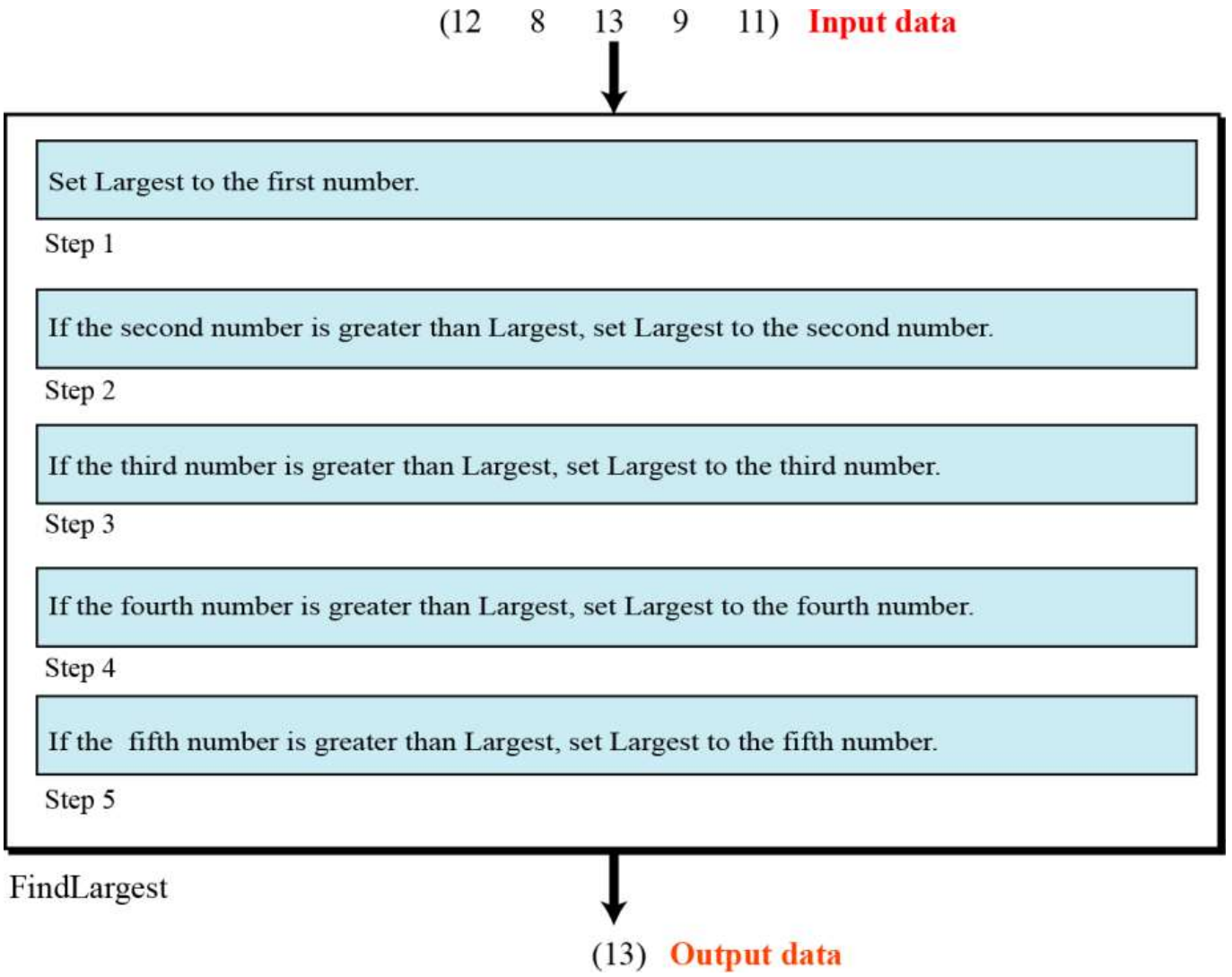
- ***Solution***

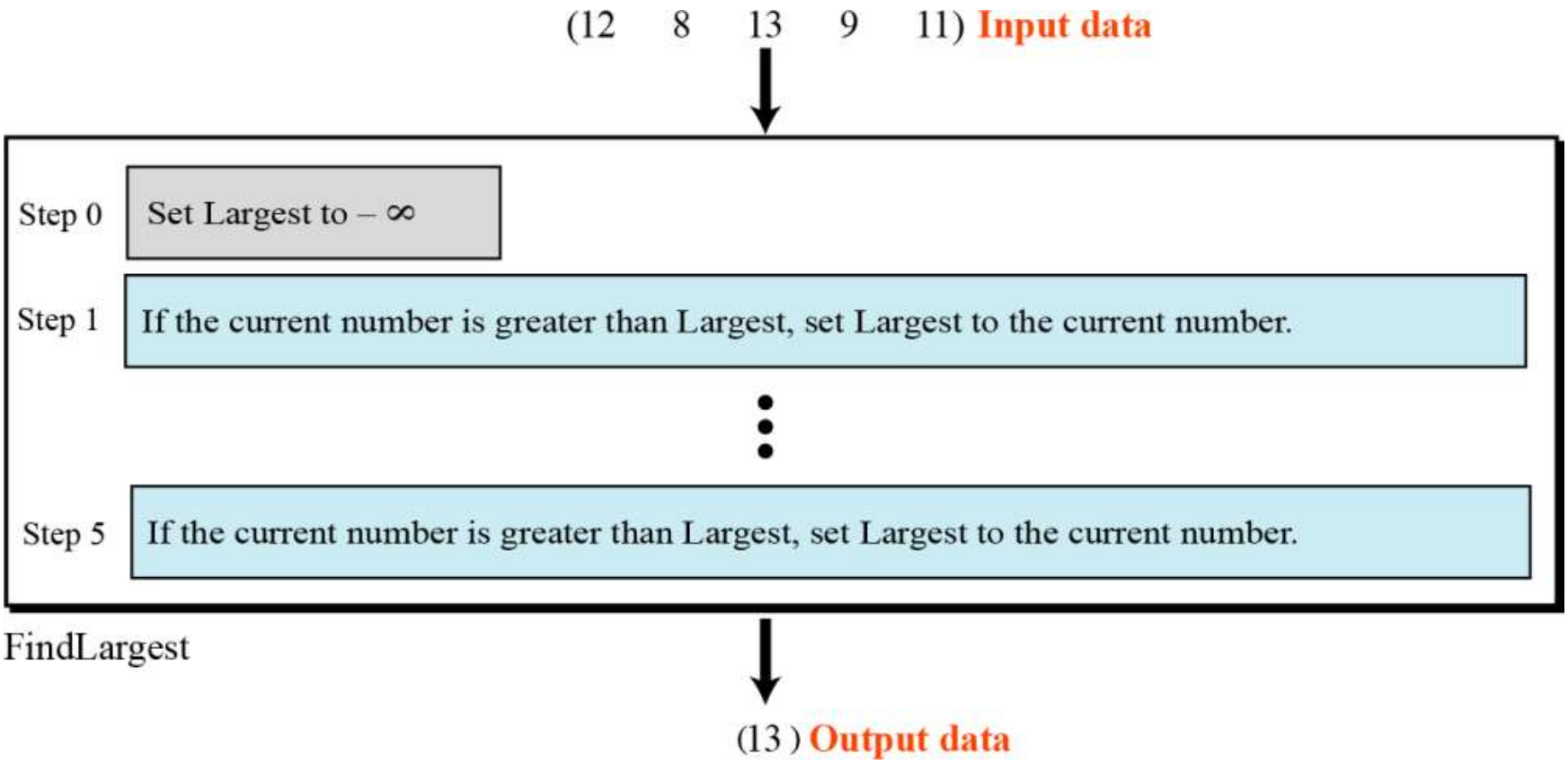
a) To solve this problem, we need an intuitive approach

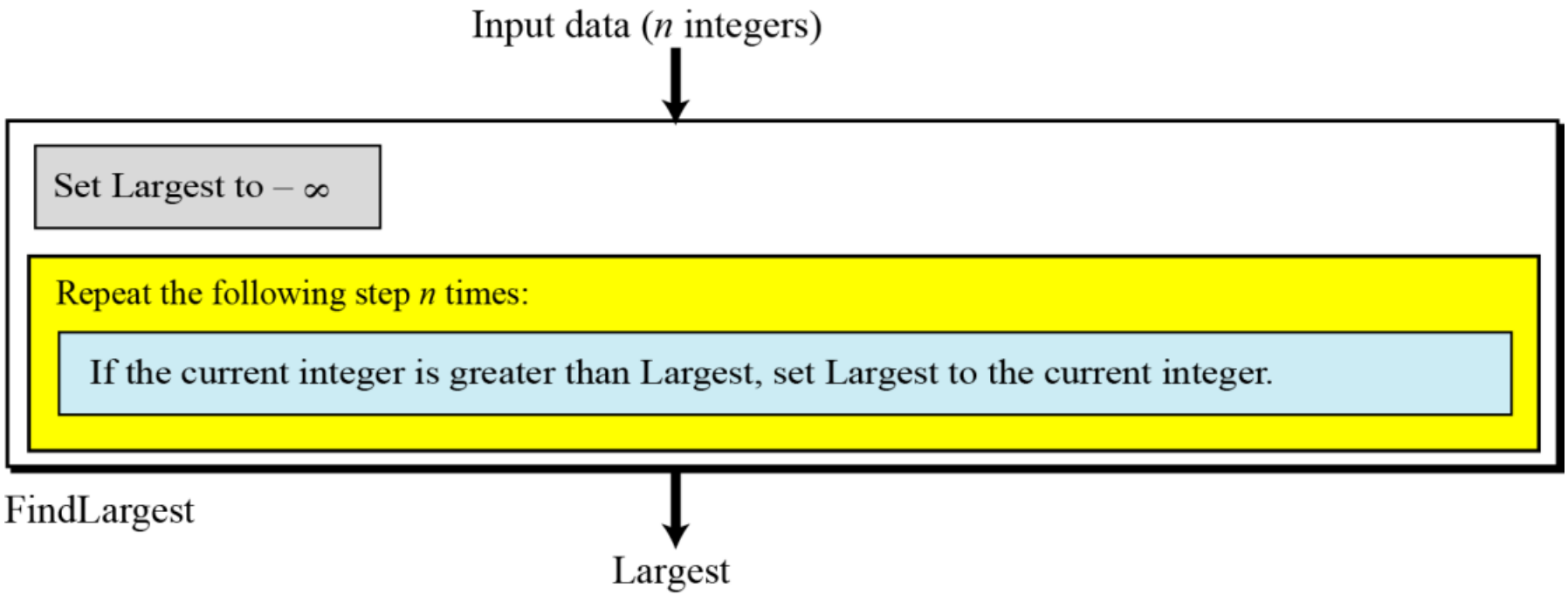
b) First use a small number of integers (for example, five), then extend the solution to any number of integers

c) The algorithm receives a list of five integers as input and gives the largest integer as output


(12 8 13 9 11) Input data
Largest 12 12 8 13 9 11 List
Step 1
Largest 12 12 8 13 9 11 List
Step 2
Largest 13 12 8 13 9 11 List
Step 3
Largest 13 12 8 13 9 11 List
Step 4
Largest 13 12 8 13 9 11 List
Step 5
FindLargest
(13) Output data


(12 8 13 9 11) Input data
Set Largest to the first number.
Step 1
If the second number is greater than Largest, set Largest to the second number.
Step 2
If the third number is greater than Largest, set Largest to the third number.
Step 3
If the fourth number is greater than Largest, set Largest to the fourth number.
Step 4
If the fifth number is greater than Largest, set Largest to the fifth number.
Step 5
FindLargest
(13) Output data


(12 8 13 9 11) Input data
Step 0
Set Largest to – ∞
Step 1
If the current number is greater than Largest, set Largest to the current number.
Step 5
If the current number is greater than Largest, set Largest to the current number.
FindLargest
(13) Output data


Input data (n integers)
Set Largest to – ∞
Repeat the following step n times:
If the current integer is greater than Largest, set Largest to the current integer.
FindLargest
Largest

Creating Algorithms Contd…

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

Creating Algorithms Contd…

Example 3

Convert Temperature from Fahrenheit (℉) to Celsius (℃)

- **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

Creating Algorithms Contd…

Example 4

Algorithm to Add Two Numbers Entered by User

- **Step 1:** Start
- **Step2:** 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.

  sum←num1+num2
- **Step 5:** Display sum Step 6: Stop

Creating Algorithms Contd…

- **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

Writing Pseudocode

- Pseudo – Imitation / False
- Code – Instructions
- **Goal:** To provide a high level description of the Algorithm
- **Benefit:** Enables programmer to concentrate on Algorithm
- Similar to programming code
- Description of the Algorithm
- No specific Programming language notations
- Pseudo Code transformed into actual program code

Writing Pseudocode Contd…

a) Guidelines for Writing Pseudo Code

- **Write only one Statement per line**
- Example – Pseudo Code for calculating Salary

1. **READ** name, hourly rate, hours worked, deduction rate
2. Gross pay = hourly rate * hours worked
3. deduction = gross pay * deduction rate
4. net pay = gross pay – deduction
5. **WRITE** name, gross, deduction, net pay

Writing Pseudocode Contd…

b) Capitalize Initial Keyword

- Keywords to be written in capital letters
- Examples: **READ, WRITE, IF, ELSE, WHILE, REPEAT, PRINT**

c) Indent to show Hierarchy

- Indentation shows the structure boundaries
- Sequence
- Selection
- Looping

Writing Pseudocode Contd…

d) End Multiline structures

- Each structure must end properly
- Example: IF statement must end with ENDIF

e) Keep Statements Language independent

- Resist the urge to write Pseudo Code in any programming language

Writing Pseudocode Contd…

- **Advantages**
  - Easily typed in a Word document
  - Easily modified
  - Simple to Use and understand
  - Implements Structured Concepts
  - No special symbols are used
  - No specific syntax is used
  - Easy to translate into Program

Writing Pseudocode Contd…

- **Disadvantages**
  - No accepted Standard
  - Cannot be compiled and executed

Writing Pseudocode Contd...

- **Write an Pseudo Code to:**

1) Add three numbers and Display the result
2) Calculate Sum and product of two numbers
3) Input examination marks and award grades according to the following criteria:
   a) > = 80 Distinction
   b) > = 60 First Class
   c) > = 50 Second Class
   d) < 40 Fail

Writing Pseudocode Contd…

1. Pseudo Code to Add Three Numbers

- Use Variables: sum, num1, num2, num3 of type integer
- ACCEPT num1,num2,num3
- Sum = num1+num2+num3
- Print sum
- End Program

Writing Pseudocode Contd…

1. **Calculate Sum and product of two numbers**
   - Use Variables: sum, product, num1, num2 of type real
   - DISPLAY “Input two Numbers”
   - ACCEPT num1,num2
   - Sum = num1+num2
   - Print “The sum is”, sum
   - product = num1*num2
   - Print “The product is”, product
   - End Program

Writing Pseudocode Contd…

3. **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

Single Line and Multiline Comments

- **Comment – Definition**
  - Used to provide information about lines of code
  - Provide clarity to the C source code
  - Allows others to better understand what the code was intended to
  - Helps in debugging the code
  - Important in large projects containing hundreds or thousands of lines of source code
  - Types – Single line and multiline comment

Single Line and Multiline Comments Contd…

a) Single Line Comment

- Represented by double slash \\

```
#include<stdio.h>
int main( ){
   //printing information
   printf("Hello C");
return 0;
}
```

Single Line and Multiline Comments Contd…

b) Multi-Line Comment

- Represented by slash asterisk * … *\

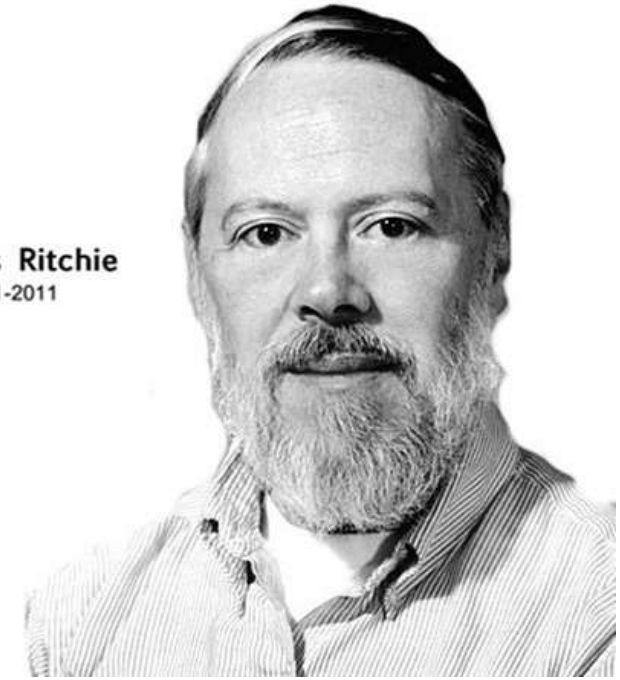
```
#include<stdio.h>
int main( ){
   /*printing information
   Multi Line Comment*/
   printf("Hello C");
return 0;
}
```

Single Line and Multiline Comments Contd…

Single-Line Comments	Multi-Line Comment
Starts with /* and ends with */	Starts with //
All Words and Statements written between /* and */ are ignored	Statements after the symbol // upto the end of line are ignored
Comment ends when */ Occures	Comment Ends whenever ENTER is Pressed and New Line Starts
e.g /* using for loop print welcome to pps class for 10 time */	e.g // Program for Fibonacci

History & Evolution of C

- C – General Purpose Programming Language
- Developed by Dennis Ritchie in 1972
- Developed at Bell Laboratories
- Principles taken from BCPL and CPL
- Structured Programming Language
- C Program
  - Collection of Functions
  - Supported by C library



History & Evolution of C Cont...

Father of C Programming : Dennis Ritchie

Born On	September 9 1941
Born in	Bronxville – New York
Full Name	Dennis MacAlistair Ritchie
Nickname	DMR
Nationality	American
Graduate From	Harvard University
Graduate In	Physics and Applied Mathematics
Webpage	http://cm.bell-labs.com/who/dmr/
Dead On	October 12 2011

History & Evolution of C Cont...

1960	Algol	• International Group
1967	BCPL	• Martin Richards
1970	B	• Ken Thomson
1972	Traditional C	• Dennis Ritchie
1978	K&R C	• kernighan & Ritchie
1989	ANSI C	• ANSI Commitee
1990	ANSI/ISO C	• ISO Commitee
1999	C99	• Standerd Commitee

Evolution of C

History & Evolution of C Cont...

- ***Why the Name "C" was given ?***
  - Many of C's principles and ideas were derived from the earlier language B
  - BCPL and CPL are the earlier ancestors of B Language (CPL is common Programming Language)
  - In 1967, BCPL Language (Basic CPL) was created as a scaled down version of CPL
  - As many of the **features were derived from "B" Language the new language was named as "C"**.

History & Evolution of C Cont…

- **Characteristics of 'C'**
  - Low Level Language Support
  - Structured Programming
  - Extensive use of Functions
  - Efficient use of Pointers
  - Compactness
  - Program Portability
  - Loose Typing

History & Evolution of C Cont…

- **Advantages of C**
  - Compiler based Language
  - Programming – Easy & Fast
  - Powerful and Efficient
  - Portable
  - Supports Graphics
  - Supports large number of Operators
  - Used to Implement Datastructures

History & Evolution of C Cont...

- **Disadvantages of C**
  - Not a strongly typed Language
  - Use of Same operator for multiple purposes
  - Not Object Oriented

Structure of 'C' Program

- Structure based on Set of rules defined by the Compiler
- **Sections**

1) Documentation
2) Preprocessor
3) Global Declaration
4) main( ) function
5) Local Declaration
6) Program Statements

Structure of 'C' Program Contd…

❑ Rules for Writing a C Program

a) All statements should be written in lower case

b) All statements should end with a semicolon

c) Upper case letters are used for symbolic constants

d) Blank spaces can be inserted between words

e) No blank space while declaring a variable, keyword, constant

f) Can write one or more statement in same line separated by comma

g) Opening and closing of braces should be balanced

Structure of 'C' Program Contd…

Documentation section

Link section

Definition section

Global declaration section

main () Function section

{

Declaration part
Executable part

}

Structure of 'C' Program Contd…

```
/* Program to Find Area of Circle */
```
Comment

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

```
const float pi = 3.14;
```
Global Declaration

```
void main( )
```
main Function

```
{
    float area;
    int r;
```
Local Declaration & Initialization

```
    printf("Enter the Radius of the Circle");
    scanf("%d", &r);
    area = pi * r * r;
    printf("The area of the Circle is %f", area);
    getch( );
}
```
Execution

Structure of 'C' Program Contd…

1) Documentation Section

- Used for providing Comments
- Comment treated as a single white space by Compiler
- Ignored at time of Execution: Not Executable
- Comment: Sequence of Characters given between /* and */
- **Example:** Program Name, Statement description

/* Program to Find Area of a Circle*/

Structure of 'C' Program Contd…

2) Preprocessor Section

- Also called as Preprocessor Directive
- Also called as Header Files
- Not a part of Compiler
- Separate step in Compilation Process
- Instructs Compiler to do required Preprocessing
- Begins with # symbol
- Preprocessor written within < >

Structure of 'C' Program Contd…

- **Examples**
  - #include <stdio.h>
  - #include <conio.h>
  - #include <math.h>
  - #include <stdlib.h>
  - #define PI 3.1412

Structure of 'C' Program Contd…

Directive	Description
#define	Substitutes a preprocessor macro.
#include	Inserts a particular header from another file.
#undef	Undefines a preprocessor macro.
#ifdef	Returns true if this macro is defined.
#ifndef	Returns true if this macro is not defined.
#if	Tests if a compile time condition is true.
#else	The alternative for #if.
#elif	#else and #if in one statement.
#endif	Ends preprocessor conditional.

Structure of 'C' Program Contd…

Directive	Description
#error	Prints error message on stderr.
#pragma	Issues special commands to the compiler, using a standardized method.

Structure of 'C' Program Contd…

3) Global Declaration Section

- Used to Declare Global variable (or) Public variable
- Variables are declared outside all functions
- Variables can be accessed by all functions in the program
- Same variable used my more than one function

Structure of 'C' Program Contd…

4) main( ) Section

- main( ) written in all small letters (No Capital Letters)
- Execution starts with a Opening Brace : {
- Divided into two sections: Declaration & Execution
  - **Declaration :** Declare Variables
  - **Executable:** Statements within the Braces
- Execution ends with a Closing Brace : }
- ***Note:*** main( ) does not end with a semicolon

Structure of 'C' Program Contd…

5) Local Declaration Section

- Variables declared within the main( ) program
- These variables are called Local Variables
- Variables initialized with basic data types

SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

Input and Output Functions

- Ability to Communicate with Users during execution
- **Input Operation**
  - Feeding data into program
  - Data Transfer from Input device to Memory
- **Output Operation**
  - Getting result from Program
  - Data Transfer from Memory to Output device
- Header File : #include**<stdio.h>**

Input and Output Functions Contd…

- **Input / Output Function Types**
  a) Formatted Input / Output Statements
  b) Unformatted Input / Output Statements

Console Input / Output Functions

Formatted Functions

Type	Input	Output
Char	scanf( )	printf( )
Int	scanf( )	printf( )
Float	scanf( )	printf( )
String	scanf( )	printf( )

Unformatted Functions

Type	Input	Output
char	getch( ) getche( ) getchar( )	putch( ) putchar( )
int	-	-
float	-	-
string	gets( )	puts( )

Input and Output Functions Contd…

a) Formatted Input / Output Statements

- Reads and writes all types of data values
- Arranges data in particular format
- Requires **Format Specifier** to identify Data type
- Basic Format Specifiers
  - %d – Integer
  - %f – Float
  - %c – Character
  - %s - String

Input and Output Functions Contd…

i. The scanf ( ) Function

- Reads all types of input data
- Assignment of value to variable during Runtime

- **Syntax**

scanf("Control String / Format Specifier", &arg1, &arg2,… &argn)

- Control String / Format Specifier
- arg1, arg2.,,, arg n – Arguments (Variables)
- & - Address

Input and Output Functions Contd…

/* Giving Direct Input in Program */

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a;
    a=10;
}
```

/*Getting Input using scanf ( ) function */

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a;
    scanf("%d", &a);
}
```

Input and Output Functions Contd…

/* Getting Multiple Input using scanf ( ) function */

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a, b, c;
    scanf("%d%d%d",&a,&b,&c);
}
```

/* Getting Multiple Different Inputs using scanf ( ) function */

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a, b;
    float c;
    scanf("%d%d%f",&a,&b,&c);

}
```

Input and Output Functions Contd…

```
/* Getting Multiple Input using scanf ( ) function */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a, b;
    float c;
    scanf("%d %d", &a, &b);
    scanf("%f ", &c);
}
```

Input and Output Functions Contd…

ii. The printf ( ) Function

- To print Instructions / Output onto the Screen
- Requires Format Specifiers & Variable names to print data

- **Syntax**

printf("Control String/Format Specifier",arg1,arg2,… argn)

- Control String / Format Specifier
- arg1, arg2.,,, arg n – Arguments (Variables)

Input and Output Functions Contd…

```
/* Example 1 – Using printf ( ) & scanf ( ) function */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a;
    printf("Enter the Value of a");
    scanf("%d", &a);
    printf("Value of a is %d", a);
    getch( );
}
```

Input and Output Functions Contd…

```
/* Example 2 – Using printf ( ) & scanf ( ) function */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a, b, c;
    printf("Enter the Value of a, b & c");
    scanf("%d %d %d", &a, &b, &c);
    printf("Value of a, b & c is %d%d%d", a, b, c);
    getch ( );
}
```

```
/* Example 3 – Using printf ( ) & scanf ( ) function */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a, b;
    float c;
    printf("Enter the Value of a & b");
    scanf("%d %d", &a, &b);
    printf("Enter the Value of a & b");
    scanf("%f ", &c);
    printf("Value of a, b is %d%d", a, b);
    printf("Value of c is %f", c);
    getch ( );
}
```

Input and Output Functions Contd…

```
/* Example 4 – Using printf ( ) & scanf ( ) function */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a, b;
    float c;
    printf("Enter the Value of a, b & c");
    scanf("%d %d%f", &a, &b, &c);
    printf("Value of a, b & c is %d%d%f", a, b, c);
    getch ( );
}
```

Input and Output Functions Contd…

- **Try it Out Yourself ! Write a C program to:**

1) Add two numbers
2) To Multiply two floating point numbers
3) To compute Quotient and Remainder
4) To Swap two numbers

Input and Output Functions Contd…

b) Unformatted Input / Output Statements

- Works only with Character Data type
- No need of Format Specifier
- **Unformatted Input Statements**
  - i. **getch ( )** – Reads alphanumeric characters from Keyboard
  - ii. **getchar ( )** – Reads one character at a time till enter key is pressed

Input and Output Functions Contd…

iii. **gets ( )** – Accepts any string from Keyboard until Enter Key is pressed

- **Unformatted Output Statements**

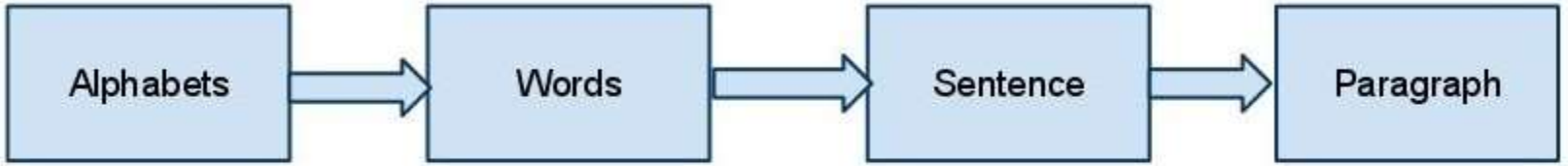
i. **putch ( )** – Writes alphanumeric characters to Monitor (Output Device)

ii. **putchar ( )** – Prints one character at a time

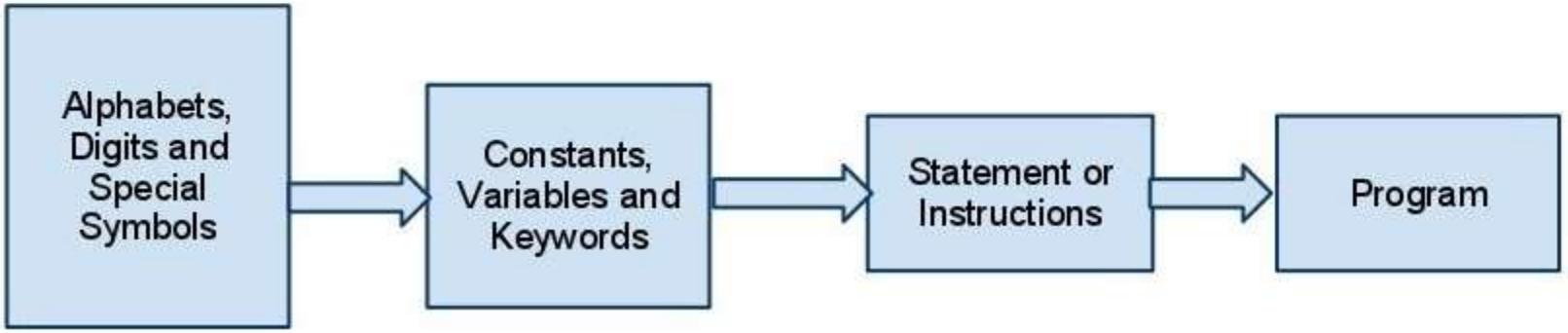
iii. **puts ( )** – Prints a String to Monitor (Output Device)

C Programming Fundamentals

Steps in Learning English Language



Steps in Learning C

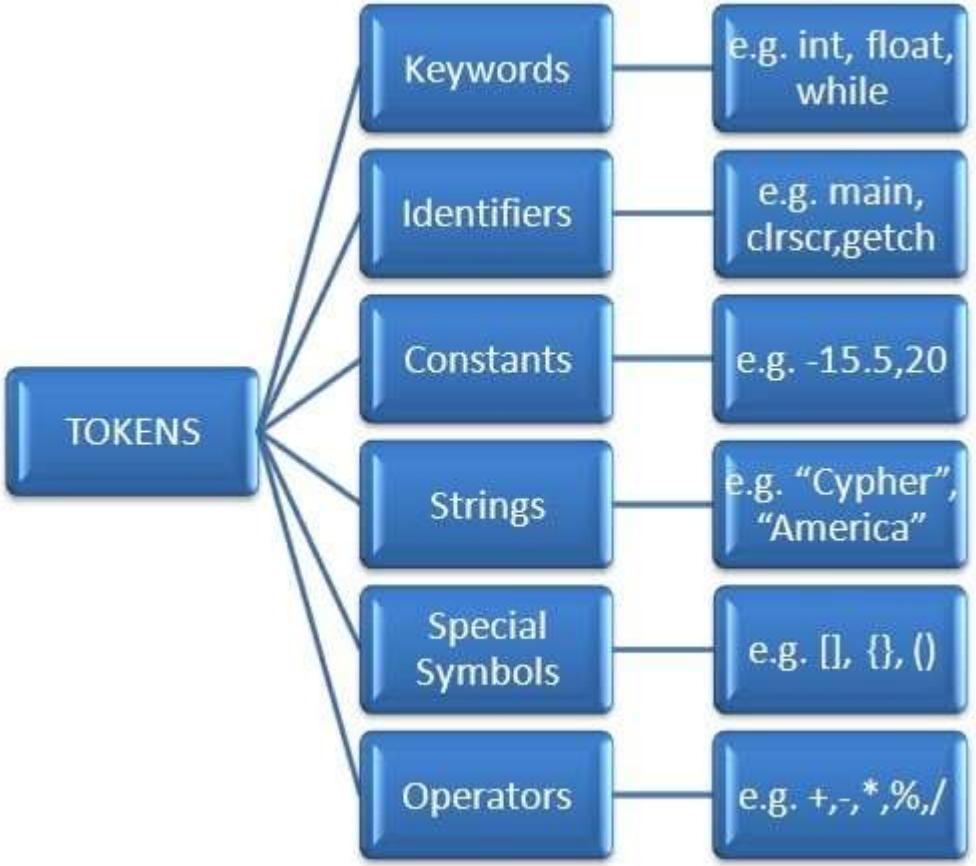


C Programming Fundamentals Contd...

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

C Programming Fundamentals Contd...

- **C Token** - Smallest individual unit of a C program
- C program broken into many C tokens
- Building Blocks of C program



Keywords

- **Keywords** – Conveys special meaning to Compiler
- Cannot be used as variable names

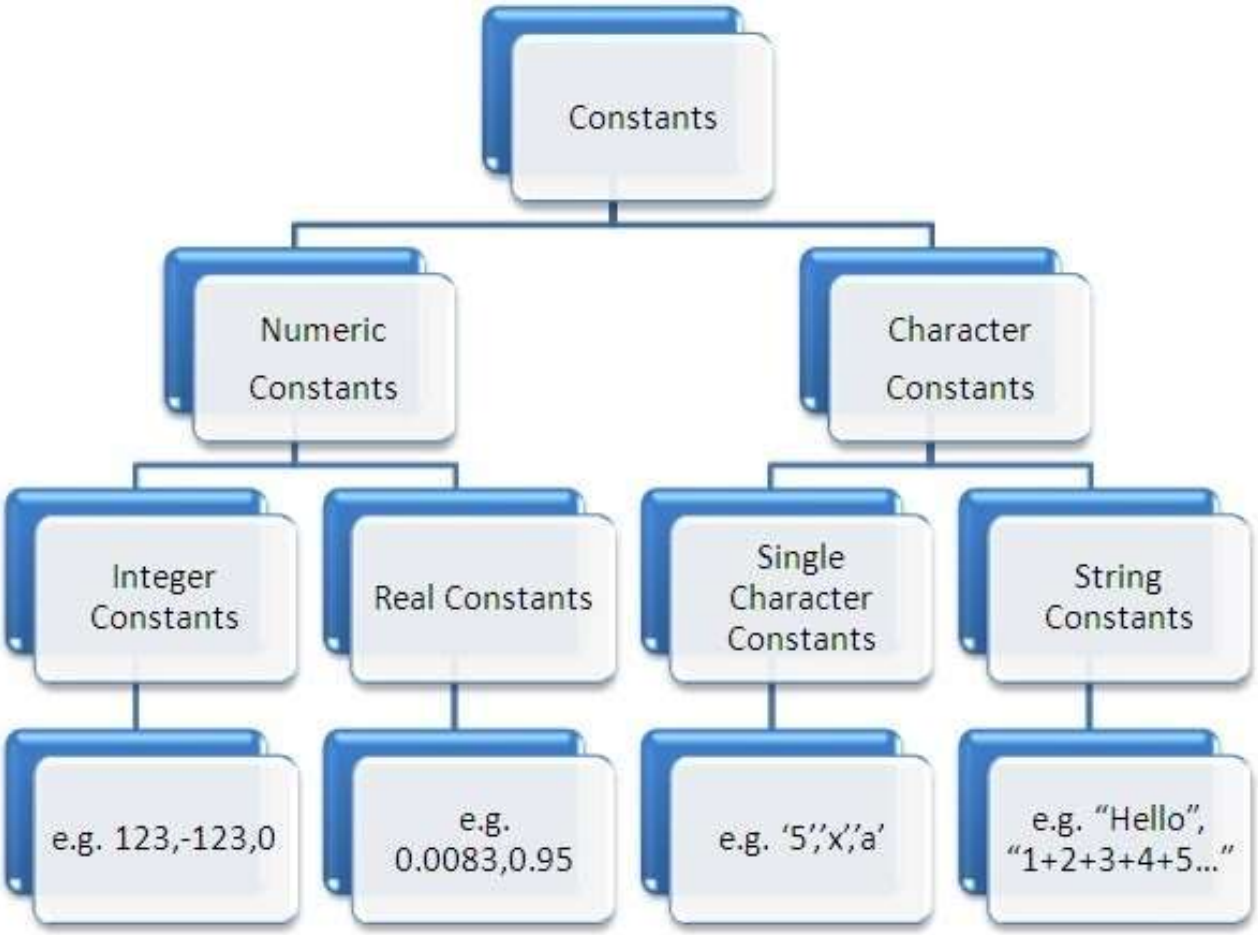
auto	**double**	**int**	**struct**
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	

SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI

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 Contd…



Variables & Identifiers

- ***Identifier***
  - A string of alphanumeric characters that begins with an alphabetic character or an underscore character
  - There are 63 alphanumeric characters, i.e., 53 alphabetic characters and 10 digits (i.e., 0-9)
  - Used to represent various programming elements such as variables, functions, arrays, structures, unions
  - The underscore character is considered as a letter in identifiers (Usually used in the middle of an identifier)

Variables & Identifiers Contd…

- **Rules for Identifers**
  - Combination of alphabets, digits (or) underscore
  - First character should be a Alphabet
  - No special characters other than underscore can be used
  - No comma / spaces allowed within variable name
  - A variable name cannot be a keyword
  - Variable names are case sensitive
- ***Variable Definition*** :Value changes during execution
- Identifier for a memory location where data is stored

Variables & Identifiers Contd…

- Variable name length cannot be more than 31 characters
- **Examples:** AVERAGE, height, a, b, sum, mark_1, gross_pay

- **Variable Declaration**
  - A variable must be declared before it is used
  - Declaration consists of a data type followed by one or more variable names separated by commas.
  - Syntax

```
datatype variablename;
```

Variables & Identifiers Contd…

- Examples

  int a, b, c, sum;

  float avg;

  char name;

- **Variable Initialization**
  - Assigning a value to the declared variable
  - Values assigned during declaration / after declaration

Variables & Identifiers Contd…

- Examples
  - *i.* *int a, b, c;*

    a=10, b=20, c=30;
  - *ii.* *int a=10 ,b=10, c=10;*
- **Scope of Variables**
- Local Variables
- Global Variables

Scope of Variables

- ***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 Variables Contd…

- 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 Variables Contd…

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

/* 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 Variables Contd…

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

/* 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;
}
```

Scope of Variables Contd…

- ***Note:*** A program can have same name for local and global variables but the value of local variable inside a function will take preference

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 Contd…

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 Contd…

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 in C

- What is a Variable?
- Storage Class Specifiers tells the Compiler about the following:
  - Where to Store a Variable
  - What is the Initial value of the Variable
  - What is the Lifetime of a Variable
- **Variable Scope:** Area or block where the variables can be accessed

a) Automatic Variables b) External Variables

c) Static Variables d) Register variables

Storage Classes in C Contd…

a) Automatic Variable (Or) Auto Variable (Or) Local Variable

- Default Storage class
- Defined inside a Function
- **Scope of the Variable:** Local to the function block where the variable is defined
- Lifetime of the variable's content vanishes after execution
- Keyword ***Auto*** used to Erase content of the variable

```
/* Program to Demonstrate Automatic (Or) Local Variables*/
#include<stdio.h>
#include<conio.h>
void main( )
{
      int n = 10;
      block1( );
      block2( );
      printf("In Main Block n=%d", n);
      getch( );
}
block1( )
{
      int n = 20;
      printf("In Block 1 n=%d", n);
}
```

```
block2( )
{
    int n = 30;
    printf("In Block 2 n=%d", n);
}
```

Output

In Block 1 n=20

In Block 2 n= 30

In Main Block n=10

Storage Classes in C Contd…

b) External Variable (Or) Global Variable

- Available to all the Functions
- Defined outside the Function
- Keyword ***Declare*** is used to define the global variable (Optional)
- **Scope of the Variable:** Global to all the function blocks
- Lifetime of the variable's content vanishes after the entre program is executed

```
/* Program to Demonstrate External / Global Variables*/
#include<stdio.h>
#include<conio.h>
int n = 10;
void main( )
{
      block1( );
      block2( );
      clrscr( );
      printf("In Main Block n=%d", n);
      getch( );
}
block1( )
{
      printf("In Block 1 n=%d", n);
      return;
}
```

```
block2( )
{
    printf("In Block 2 n=%d", n);
    return;
}
```

Output

```
In Block 1 n=10

In Block 2 n= 10

In Main Block n=10
```

Storage Classes in C Contd…

c) Static Variables

- Keyword ***Static*** is used to define the variable (Compulsory)
- Variable declared as static is initialized to NULL
- Value of the Static variable remains the same throughout the program
- **Scope of the Variable:** Local or Global depending on where it is declared
- **Static Global:** Defined outside the Function
- **Static Local:** Defined inside the Function

```
/* Program to Demonstrate Static Variables*/
#include<stdio.h>
#include<conio.h>
void main( )
{
      int x;
      static int y;
      clrscr( );
      printf("x=%dy=%d", x, y);
      getch( );
}
```

Output

x = 28722

y = 0

Storage Classes in C Contd…

d) Register Variables

- Variables stored in the CPU registers instead of Memory
- Keyword ***Register*** is used to define the variable
- **Scope of the Variable:** Local
- **Advantages**
  - CPU register access is faster than memory access
- **Disadvantages**
  - Number of CPU registers is less
  - Less Number of variables can be stored in CPU registers

```
/* Program to Demonstrate Register Variables*/
#include<stdio.h>
#include<conio.h>
void main( )
{
    register int n=1;
    clrscr( );
    for(n=1; n<=10; n++)
    printf("%d", n);
    getch( );
}
```

Output

1 2 3 4 5 6 7 8 9 10

Datatypes

- Defines a variable before use
- Specifies the type of data to be stored in variables
- Basic Data Types – 4 Classes
  a) int – Signed or unsigned number
  b) float – Signed or unsigned number having Decimal Point
  c) double – Double Precision Floating point number
  d) char – A Character in the character Set
- Qualifiers

Datatypes Contd…

Variable Type	Keyword	Bytes Required	Range	Format
Character (signed)	Char	1	-128 to +127	%c
Integer (signed)	Int	2	-32768 to +32767	%d
Float (signed)	Float	4	-3.4e38 to +3.4e38	%f
Double	Double	8	-1.7e308 to + 1.7e308	%lf
Long integer (signed)	Long	4	2,147,483,648 to 2.147.438.647	%ld
Character (unsigned)	Unsigned char	1	0 to 255	%c
Integer (unsigned)	Unsigned int	2	0 to 65535	%u
Unsigned long integer	unsigned long	4	0 to 4,294,967,295	%lu
Long double	Long double	10	-1.7e932 to +1.7e932	%Lf

Datatypes Contd…

a) Integer Data Type

- Whole numbers with a range
- No fractional parts
- Integer variable holds integer values only
- **Keyword:** int
- **Memory:** 2 Bytes (16 bits) or 4 Bytes (32 bits)
- **Qualifiers:** Signed, unsigned, short, long
- **Examples:** 34012, 0, -2457

Datatypes Contd…

a) Integer Data Type

```
#include<stdio.h>
int main()
{
  int a;
scanf("%d",&a);
printf("%d",a);
return 0;}
```

Datatypes Contd…

b) Floating Point Data Type

- Numbers having Fractional part
- Float provides precision of 6 digits
- Integer variable holds integer values only
- **Keyword:** float
- **Memory:** 4 Bytes (32 bits)
- **Examples:** 5.6, 0.375, 3.14756

Datatypes Contd…

b) Floating Point Data Type

```
#include<stdio.h>
int main()
{
 float g;
scanf("%f",&g);
printf("%f",g);
return 0;
}
```

Datatypes Contd…

c) Double Data Type

- Also handles floating point numbers
- Double provides precision of 14 digits
- Integer variable holds integer values only
- **Keyword:** double
- **Memory:** 8 Bytes (64 bits) or 10 Bytes (80 bits)

Datatypes Contd…

c) Double Data Type

```
#include<stdio.h>
int main()
{
   double g;
  scanf(“%ld”,&g);
printf(“%ld”,g);
return 0;
}
```

Datatypes Contd…

d) Character Data Type

- handles one character at a time
- **Keyword: char**
- **Memory:** 1 Byte (8 bits)

Datatypes Contd…

d) Character Data Type

```
#include<stdio.h>
int main()
{
 char g;
scanf("%c",&g);
printf("%c",g);
return 0;
}
```

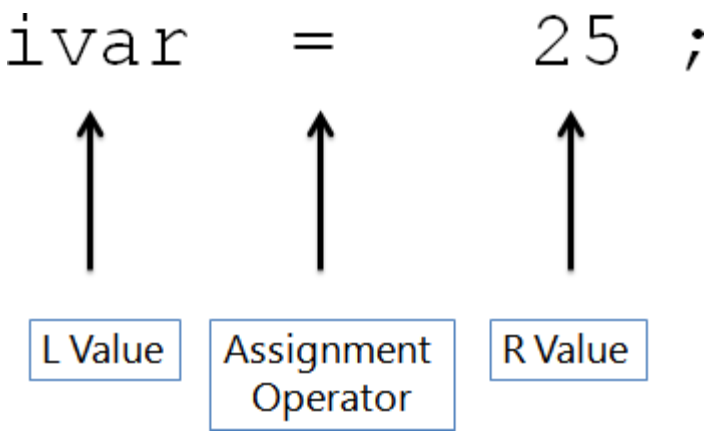
L-Value and R-Value of Expression

a) 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

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

- Diagram Showing L-Value of Expression :



SRM
INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

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

SRM
INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

L-Value and R-Value of Expression Contd…

```
#include<stdio.h>
#define MAX 20
int main( )
{
    MAX = 20; //Error
    return(0);
}
```

L-value cannot be a MACRO

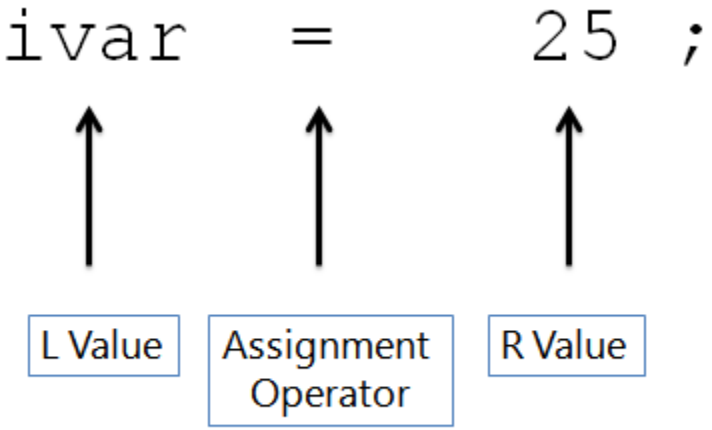
```
#include<stdio.h>
enum {JAN,FEB,MARCH};
int main( )
{
    JAN = 20; //Error
    return(0);
}
```

L-value cannot be a Enum Constant

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

b) R Value stands for **Right value** of the expression

- In any **Assignment statement** R-Value of Expression must be anything which is capable of returning Constant Expression or Constant Value



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

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

Expressions

- **Expression :** An Expression is a collection of operators and operands that represents a specific value
- **Operator** : A symbol which performs tasks like arithmetic operations, logical operations and conditional operations
- **Operands :** The values on which the operators perform the task
- Expression Types in C
    a) Infix Expression
    b) Postfix Expression
    c) Prefix Expression

Expressions Contd…

a) *Infix Expression*

- The operator is used between operands
- ***General Structure :*** Operand1 Operator Operand2
- ***Example :*** a + b

b) *Postfix Expression*

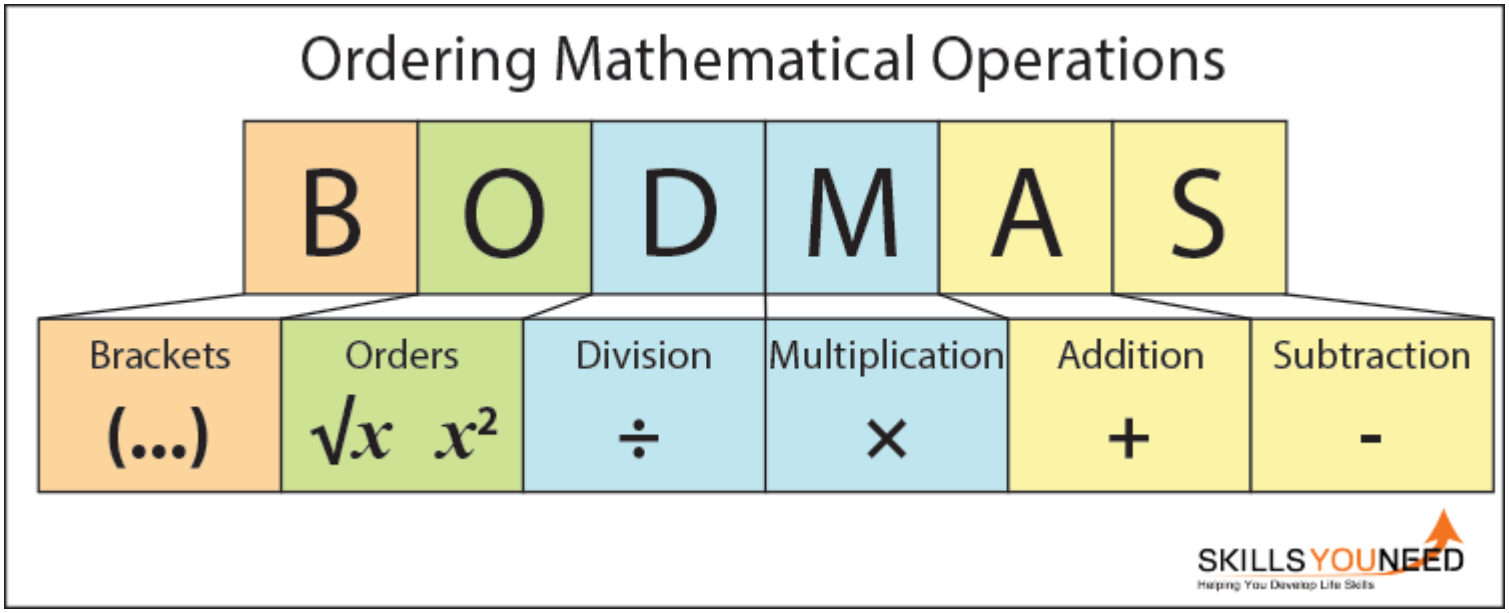
- Operator is used after operands
- ***General Structure :*** Operand1 Operand2 Operator
- ***Example :*** ab+

Expressions Contd…

c) Prefix Expression

- Operator is used before operands
- ***General Structure :*** Operator Operand1 Operand2
- ***Example :*** +ab

Expressions Contd…



SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

Precedence order	Operator	Associativity
1	( ) [ ] →	Left to right
2	++ -- - (unary) ! ~ * & sizeof	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	<< >>	Left to right
6	< <= > >=	Left to right
7	== !=	Left to right
8	& (bitwise AND)	Left to right
9	^ (bitwise XOR)	Left to right
10	| (bitwise OR)	Left to right

SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

Operators in C

- C supports rich set of built in Operators
- Used to manipulate Constants (Data) & Variables
- Part of Mathematical (or) Logical expressions
- Operators vs Operands
- **Operator – Definition**
  - Symbol (or) Special character that instructs the compiler to perform mathematical (or) Logical operations

Operators in C Contd…

- **Classification of Operators**
  a) Increment & Decrement Operators
  b) Comma Operator
  c) Arrow Operator
  d) Assignment Operators
  e) Bitwise Operators
  f) Sizeof Operator

Operators in C Contd...

a) Increment and Decrement Operators

- Increment and decrement operators are unary operators that add or subtract one from their operand
- C languages feature two versions (pre- and post-) of each operator
  - Operator placed before variable (Pre)
  - Operator placed before variable (Post)
- T he increment operator is written as ++ and the decrement operator is written as --

Operators in C Contd…

a) Increment and Decrement Operators Contd…

- **Classification**
  - Pre Increment Operator
  - Post Increment Operator
  - Pre Decrement Operator
  - Post Decrement Operator

Operators in C Contd…

a) Increment and Decrement Operators Contd…

- ***Syntax***

(pre)++variable_name;	*(pre)- -variable_name;*

(Or)

variable_name++ (post);	*variable_name – (Post);*

- ***Examples***
  - ++count, ++a, ++i, ++count
  - Count++, a++, i++, count++

Operators in C Contd...

a) Increment and Decrement Operators Contd...

S. No	Operator type	Operator	Description
1	Pre Increment	++i	Value of i is incremented before assigning it to variable i.
2	Post Increment	i++	Value of i is incremented after assigning it to variable i.
3	Pre Decrement	-- i	Value of i is decremented before assigning it to variable i.
4	Post Decrement	i --	Value of i is decremented after assigning it to variable i.

/* Program for Post Increment*/

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

Output

1 2 3 4

Operators in C Contd...

a) Increment and Decrement Operators Contd...

- **Step 1 :** In this program, value of i "0" is compared with 5 in while expression.
- **Step 2 :** Then, value of "i" is incremented from 0 to 1 using post-increment operator.
- **Step 3 :** Then, this incremented value "1" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "1 2 3 4 5".

```
/* Program for Pre Increment*/
#include<stdio.h>
#include<conio.h>
void main( )
{
      int i = 1;
     while (++i<5)
     {
          printf("%d", i );
     }              }
getch ( );
}
```

Output

2 3 4

Operators in C Contd...

a) Increment and Decrement Operators Contd...

- **Step 1 :** In above program, value of "i" is incremented from 0 to 1 using pre-increment operator.
- **Step 2 :** This incremented value "1" is compared with 5 in while expression.
- **Step 3 :** Then, this incremented value "1" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "1 2 3 4".

/* Program for Post Decrement*/

```
#include<stdio.h>
#include<conio.h>
void main( )
{
      int i = 10;
     while (i--<5)
     {
          printf("%d", i );
     }              }
getch ( );
}
```

Output

10 9 8 7 6

Operators in C Contd...

a) Increment and Decrement Operators Contd...

- **Step 1 :** In this program, value of i "10" is compared with 5 in while expression.
- **Step 2 :** Then, value of "i" is decremented from 10 to 9 using post-decrement operator.
- **Step 3 :** Then, this decremented value "9" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "9 8 7 6 5".

/* Program for Pre Decrement*/

```
#include<stdio.h>
#include<conio.h>
void main( )
{
      int i = 10;
     while (--i<5)
     {
          printf("%d", i);
     }              }
getch ( );
}
```

Output

9 8 7 6

Operators in C Contd...

a) Increment and Decrement Operators Contd...

- **Step 1 :** In above program, value of "i" is decremented from 10 to 9 using pre-decrement operator.
- **Step 2 :** This decremented value "9" is compared with 5 in while expression.
- **Step 3 :** Then, this decremented value "9" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "9 8 7 6".

Operators in C Contd...

b) Comma Operator

- Special operator which separates the declaration of multiple variables
- Has Lowest Precedence i.e it is having lowest priority so it is evaluated at last
- Returns the value of the rightmost operand when multiple comma operators are used inside an expression
- Acts as Operator in an Expression and as a Separator while Declaring Variables

Operators in C Contd...

b) Comma Operator Contd...

```
#include<stdio.h>
int main( )
{
int i, j;
i=(j=10, j+20);
printf("i = %d\n j = %d\n" , i,j );
return 0;
}
```

Operators in C Contd...

c) Arrow Operator (->)

- Arrow operator is used to access the structure members when we use pointer variable to access it
- When pointer to a structure is used then arrow operator is used

c) Arrow Operator (->)

```
#include <stdio.h>

#include<stdlib.h>

struct abc {

int a;};

int main(){

struct abc * g;    g=(struct abc *)malloc(sizeof(struct abc));

g->a=19;

printf("%d",g->a);

return 0;}
```

Operators in C Contd…

d) Assignment Operators

- Assigns result of expression to a variable
- Performs Arithmetic and Assignment operations
- Commonly used Assignment operator: **=**
- ***Syntax*** ***variable = expression;***
- ***Examples***
  - num = 25; age = 18; pi = 31.4; area = 3.14 * r * r;

Operators in C Contd...

- Shorthand Assignment Operators

Simple Assignment Operator	Shorthand Operator
a = a + 1	a+=1
a = a – 1	a-=1
a = a * 2	a*=2
a = a / b	a/=b
a = a % b	a%=b
c = c * (a + b)	c *= (a + b)
b = b / (a + b)	b /=(a + b)

/* Program for Assignment Operations*/

```
#include<stdio.h>
#include<conio.h>
void main( )
{
      int a;
     a = 11;
     a+ = 4;
          printf("Value of A is %d\n",a);
     a = 11;
     a- = 4;
          printf("Value of A is %d\n",a);
     a = 11;
     a* = 4;
          printf("Value of A is %d\n",a);
     a = 11; a/ = 4;
```

```
        printf("Value of A is %d\n",a);
    a = 11;
    a% = 4;
        printf("Value of A is %d\n",a);
    getch ( );
}
```

Output

Value of A is 15

Value of A is 7

Value of A is 44

Value of A is 2

Value of A is 3

Operators in C Contd...

e) Bitwise Operators

- In arithmetic-logic unit, mathematical operations like: addition, subtraction, multiplication and division are done in bit-level
- To perform bit-level operations in C programming, bitwise operators are used
- Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise NOT), ^ (XOR), << (left shift) and >> (right shift)

Operators in C Contd…

e) Bitwise Operators Contd…

Operator	Operation
&	Bitwise AND
|	Bitwise OR
~	One's Complement
>>	Shift right
<<	Shift left
^	Exclusive OR

Bitwise operators: truth table

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

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -60, i.e,. 1100 0100 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

PROGRAM TO DEMONSTRATE BITWISE OPERATIONS

```
#include <stdio.h>
 int main()
{
      unsigned int a = 60; /* 60 = 0011 1100 */
      unsigned int b = 13; /* 13 = 0000 1101 */
      int c = 0; c = a & b; /* 12 = 0000 1100 */
      printf("Line 1 - Value of c is %d\n", c ); c = a | b; /* 61 = 0011 1101 */
      printf("Line 2 - Value of c is %d\n", c ); c = a ^ b; /* 49 = 0011 0001 */
      printf("Line 3 - Value of c is %d\n", c ); c = ~a; /*-61 = 1100 0011 */
      printf("Line 4 - Value of c is %d\n", c ); c = a << 2; /* 240 = 1111 0000 */
      printf("Line 5 - Value of c is %d\n", c ); c = a >> 2; /* 15 = 0000 1111 */
      printf("Line 6 - Value of c is %d\n", c );
      return 0;
}
```

Output

```
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
```

Operators in C Contd...

f) Sizeof Operators

- This operator returns the size of its operand in bytes
- The sizeof operator always precedes its operand
  - Used to calculate the size of data type or variables
  - Can be nested
  - Returns the size in integer format
- Syntax looks more like a function but it is considered as an operator in c programming

/* Program for Sizeof Operators*/

```
#include<stdio.h>
int main( )
{
int a;
float b;
double c;
char d;
printf("Size of Integer         :%d\n\n",sizeof(a));
printf("Size of Floating Point         :%d\n\n",sizeof(b));
printf("Size of Double         :%d\n\n",sizeof(c));
printf("Size of Charcter         :%d\n\n",sizeof(d));
return 0;
}
```

Output

Size of Integer :2

Size of Floating Point : 4

Size of Double : 8

Size of Character : 1

Operators in C

a) Relational Operators

b) Logical Operators

c) Conditional Operators

Operators in C Contd…

a) Relational Operators

- Binary Operators (or) Boolean Operators
- Produces an integer result
  - **Condition True :** Integer value is 1
  - **Condition False :** Integer value is 0
- Compares
  - Values between two variables
  - Values between variables and constants

Operators in C Contd…

a) Relational Operators Contd…

- **Relational Expression / Boolean Expression :** An expression containing a relational operator

Relational Operatiors

Operatior	Operations	Example
<	Less than	a<b
>	Greater than	a>b
<=	Less than or equal to	a<=b
>=	Greater than equal to	a>=b
=	Equal to	a==b
!=	Not equal to	a!=b

Operators in C Contd…

a) Relational Operators Contd…

- Consider a = 10 and b =4. The relational expression returns the following integer values

Relational Expression	Result	Return Values
a < b	False	0
a > b	True	1
a < = b	False	0
a > = b	True	1
a = = b	False	0
a ! = b	True	1

/* Program for Relational Operations*/

```
#include<stdio.h>
int main( )
{
      int a,b;
      printf("Enter the two Values\n");
      scanf("%d%d", &a, &b);
      printf("a>b is %d\n", (a>b));
      printf("a<b is %d\n", (a<b));
      printf("a>=b is %d\n", (a>=b));
      printf("a<=b is %d\n", (a<=b));
      printf("a==b is %d\n", (a==b));
      printf("a!=b is %d\n", (a!=b));
      return 0;
}
```

Output

4

2

a > b is 1

a < b is 0

a > = b is 1

a < = b is 0

a = = b is 0

a ! = b is 1

Operators in C Contd…

b) Logical Operators

- Combines two or more relations
- Used for testing one or more conditions

SYMBOLS	*MEANINGS*
"&&"	***LOGICAL AND:*** true when all expression are T false otherwise F
"||"	***LOGICAL OR:*** true when either expression is T false when both are F
"!"	***NOT:*** negation (T → F) and vice-versa

Operators in C Contd…

b) Logical Operators Contd…

- **Logical Expression / Compound Relational Expression :**

  An expression which combines two or more relational expression

Op1	Op2	Op1 && Op2	Op1 || Op2
F (0)	F (0)	F (0)	F (0)
F (0)	T (1)	F (0)	T (1)
T (1)	F (0)	F (0)	T (1)
T (1)	T (1)	T (1)	T (1)

Operators in C Contd…

b) Logical Operators Contd…

- Consider a = 10 and b =4. The Logical expression returns the following integer values

Relational Expression	Result	Return Values
a < 5 && b > 2	True	1
a < 5 && b < 2	False	0
a >5 && b < 2	False	0
a >5 || b < 2	True	1
a <5 || b < 2	False	0
a > 5 || b < 2	True	1

/* Program for Logical Operations*/

```
#include<stdio.h>
int main( )
{
    int age,height;
    printf("Enter Age of Candidate:\n");
    scanf("%d", &age);
    printf("Enter Height of Candidate:\n");
    scanf("%d", &height);
    if ((age>=18) && (height>=5))
        printf("The Candidate is Selected");
    else
        printf("Sorry, Candidate not Selected");
    return 0;
}
```

Output 1

Enter Age of Candidate: 18

Enter Height of Candidate: 6

The Candidate is Selected

Output 2

Enter Age of Candidate: 19

Enter Height of Candidate: 4

Sorry, Candidate not Selected

Operators in C Contd…

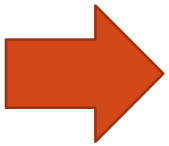
c) Conditional Operators

- ? and are the Conditional Operators
- Also called as Ternary Operators
- Shorter form of if-then-else statement
- ***Syntax***

Expression 1 ? Expression 2 : expression 3

- If expression 1 is true then the value returned will be expression 2
- Otherwise the value returned will be expression 3

```
#include<stdio.h>
int main( )
{
      int x, y;
      scanf("%d", &x);
      y=(x > 5 ? 3 : 4);
      printf("%d", y);
      return 0;
}
```



```
#include<stdio.h>
int main( )
{
      int x, y;
      scanf("%d", &x);
      if(x >5)
            y=3;
      else
            y=4;
      printf("%d", y);
      return 0;
}
```

THANK YOU