# Embedded C Programing

Trainer : Kiran Jaybhave

## Agenda

- Introduction
- C Programming
- C History
- Types of language
- VIM editor
- Hello World Program

## About Trainer

- Mr. Kiran G. Jaybhave
  - B.E. In Electronics and Telecommunication

- **Training**
  - Pre-CAT Batches
  - CDAC Main Course Batches.

- **Contact**
  - Email : kiran.jaybhave@sunbeaminfo.com

- **Daily Schedules**
  - Lecture: 8.00 am to 1.00 pm
  - Lab: 2.00 pm to 7.00 pm

- **DESD Evaluations**
  - **Each Module**
    - **Theory Exam**: MCQ : 40 marks -- CCEE
    - **Lab Exam** : Last day of the module : 40 marks
    - **Internal Evaluation** : On-going : Assignments, Quiz, Interview, etc.

## Computer System

- **What is Computer ?**
  - A **computer** is an **electronic device** that accepts input, processes it, and produces output. It works with the help of **hardware** and **software**..
  - **Hardware**: The physical, tangible parts of the computer. It performs operations and data processing.
  - **Software**: A set of instructions/programs that direct the hardware to perform specific tasks.

- **What is Program ?**
  - A **program** is a **set of instructions** written in a programming language, arranged in a logical order, to perform a specific task.
  - **A program can be constructed using various program constructs like**
    - Sequence
    - Decision Control
    - Loop / Iteration

## Classification of Languages

❑A Program can be written using

- **High Level Language**
  - Make easy communication to computer system
  - Independent to particular type of computer
  - More close to human language than machine language.
  - Compiler helps to convert instructions to machine understandable form.
  - Takes additional time to translate the instructions to machine understandable instructions

- **Low Level Language**
  - These are basic computer instructions that directly communicate with the hardware.
  - Written in binary (0 and 1) or in assembly language.
  - Machine-dependent: Specific to the processor type.
  - Very fast execution, but difficult to write and understand.

## Classification of Languages

❑**Machine Level Language**

- It is the **lowest-level programming language** consisting of instructions written in **binary form (0s and 1s)**.
- Since a computer's hardware understands only binary, instructions must be given in this format for direct execution.
    - Performance is good as we are directly writing the program on machine
    - Platform dependent
    - Difficult to program
    - Error prone

## Assembly Language

### ❑Assembly Language

- The assembly language contains some human-readable instructions (consists of numbers, symbols, abbreviations).
- The language Introduced in **1952**, it was designed to make machine programming more readable and manageable .
- As we know that computers can only understand the machine-level instructions, so we require a translator that converts the assembly code into machine code. The translator used for translating the code is known as an **assembler.**
  - Easier to understand and use and to locate errors
  - Platform dependent
  - Knowledge of hardware will help to program Machine level coding
  - Execution is slower compared to machine level language.

## History of C language

- C language was developed by **Dennis Ritchie** in 1972 at AT & T Bell Labs on PDP-11 machine.
- It was developed while porting UNIX from PDP-7 to PDP-11.
- Many features of C are inspired from B (Ken Thompson) and BCPL (Martin Richards).
- Initial release of C is referred as K & R C.

## Standardization

- C  was standardized by ANSI in 1989. This is referred as C89 .
- Standardization ensures C code to remain portable.
- Standard is revised multiple times to add new features in the language.
  - C89 – First ANSI standard
  - C90 – ANSI standard adopted by ISO
  - C99 – Added few C++ features like bool, inline, etc.
  - C11 – Added multi-threading feature.
  - C17 – Few technical corrections.

## Introduction

- High-level
- Compiled
- Procedural
- Block-Structured (control structures).
- Free form
- Typed
- Library Functions

## Features

- Data types
- Operators
- Control structures
- Functions
- Storage classes
- Pointers
- Arrays
- Strings
- Dynamic memory allocation
- Structure
- Union
- Enum
- File IO
- Preprocessor directives

## Strengths

- Low level memory access (pointers, data structures)
- Effective memory access (bitwise operators, bit-fields, unions)
- Can access OS features (functions/commands)
- Extensive library functions (math, strings, file IO, …)
- Compilers for different platforms & architectures
- Highly Readable (macros, enum, functions, …)

## Applications

- **System programming**
  - OS development
  - Device drivers
  - System utilities

- **Embedded programming**
  - ARM, AVR, PIC, etc.
  - IoT development

- **Language development**
  - Compiler development

- **Achievements (tiobe.com)**
  - In top-2 languages in last 40 years.
  - Language of year: 2019, 2017, 2008.

## VI editor

- **Linux text editors for command line.**
  - VI editor
  - Emacs editor
- **VI is world's best editor (Linux CLI).**
  - VI developed by "Bill Joy".
  - In BSD( Barkeley Soft. Distrbution) UNIX OS .
    - VIM = VI Improved.
- **VIM modes**
  - **Command mode**
    - **Default mode**
    - **Press "Esc" to go in command mode.**
  - **Edit mode (Insert)**
    - **Press "i" to go in Insert mode.**
    - Edit the file.
- **VIM Commands**

# Hello World Program

❑**Source Code**

- // Hello World program

```
#include
int main()
{
     printf("Hello World\n");
    return 0;
}
```

❑**Commands**

- **cmd$ gcc main.c  -o  main.out**
- **cmd$ ./main.out**

## Hello World

- printf() – library function

- stdio.h – header file

- main() – entry point function
  - void main() { … }
  - int main() { … }
  - int main(void) { … }
  - int main(int argc, char *argv[]) { … }
  - int main(int argc, char *argv[], char *envp[]) { … }

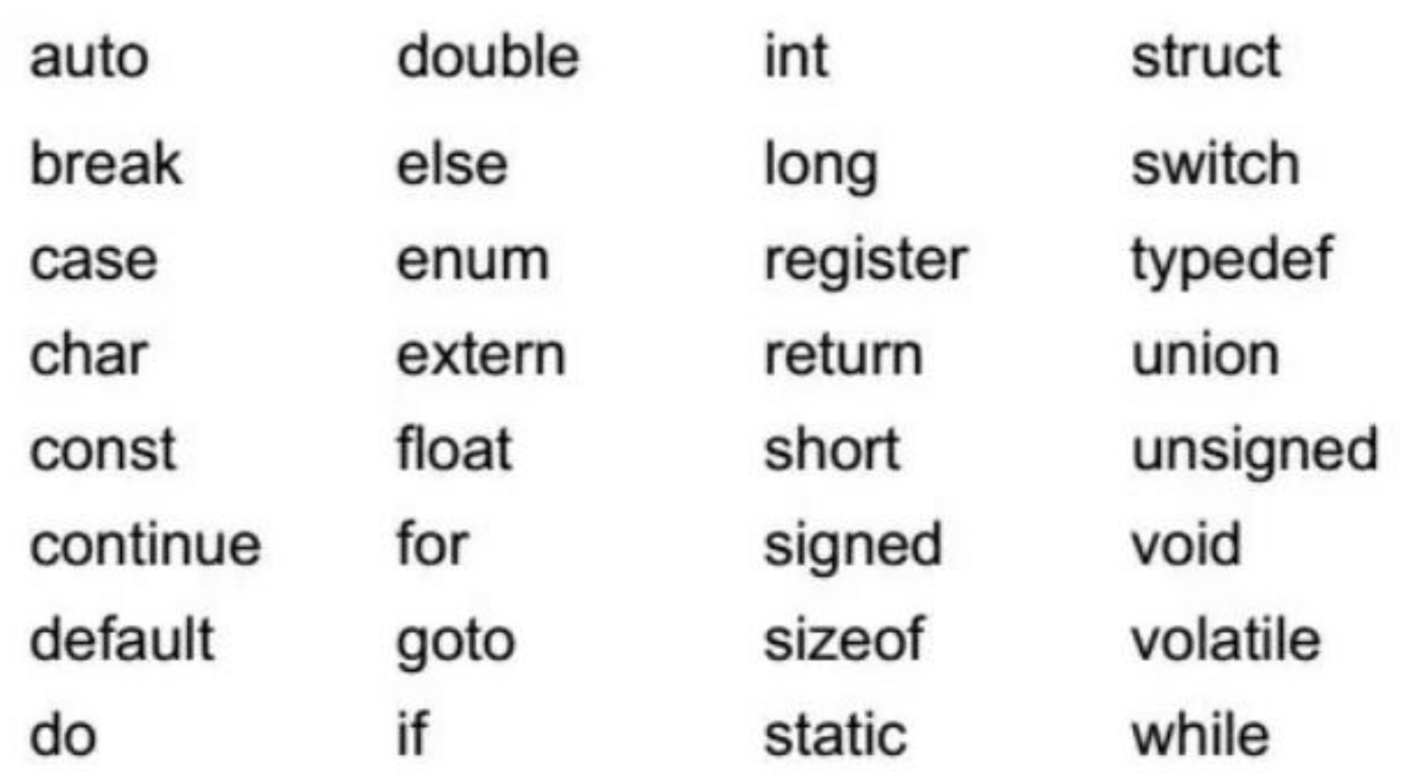
- return 0 – exit status

## Tokens

- C program is made up of functions.
- Function is made up of statements.
- Statement contain multiple tokens.
    - Keywords
    - Data Types
    - Identifiers
    - Variables
    - Constants
    - Operators

## Keywords

- Keywords are predefined words used in program, which have special meanings to the compiler.
- They are reserved words, **so cannot be used as identifier.**
- K & R C has 27 keywords.
- C89 added 5 keywords.
- C99 added 5 new keywords



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

## Data Types, Variables & Constants

- C allows computations to be performed on various types of data.
  - Numerical: Whole numbers, Real numbers
  - Character: Single character, Strings
- Fixed data values are said to be constants.
  - 12, -45, 0, 2.3, 76.9, 1.23456e+2, 'A', "Sunbeam", etc.
- Data is hold in memory locations identified by names called as variables.
  - Variable must be declared before its use in the program.
  - As per need, variable have some data type.
- Simple C data types are: **int, float , double, char.**
  - Data type represents amount of space assigned to the variable.
  - It also defines internal storage of the data.

# Data Types

- **Data type defines storage space and format of variable.**
- **Primitive types**

  **(** Format specifier )

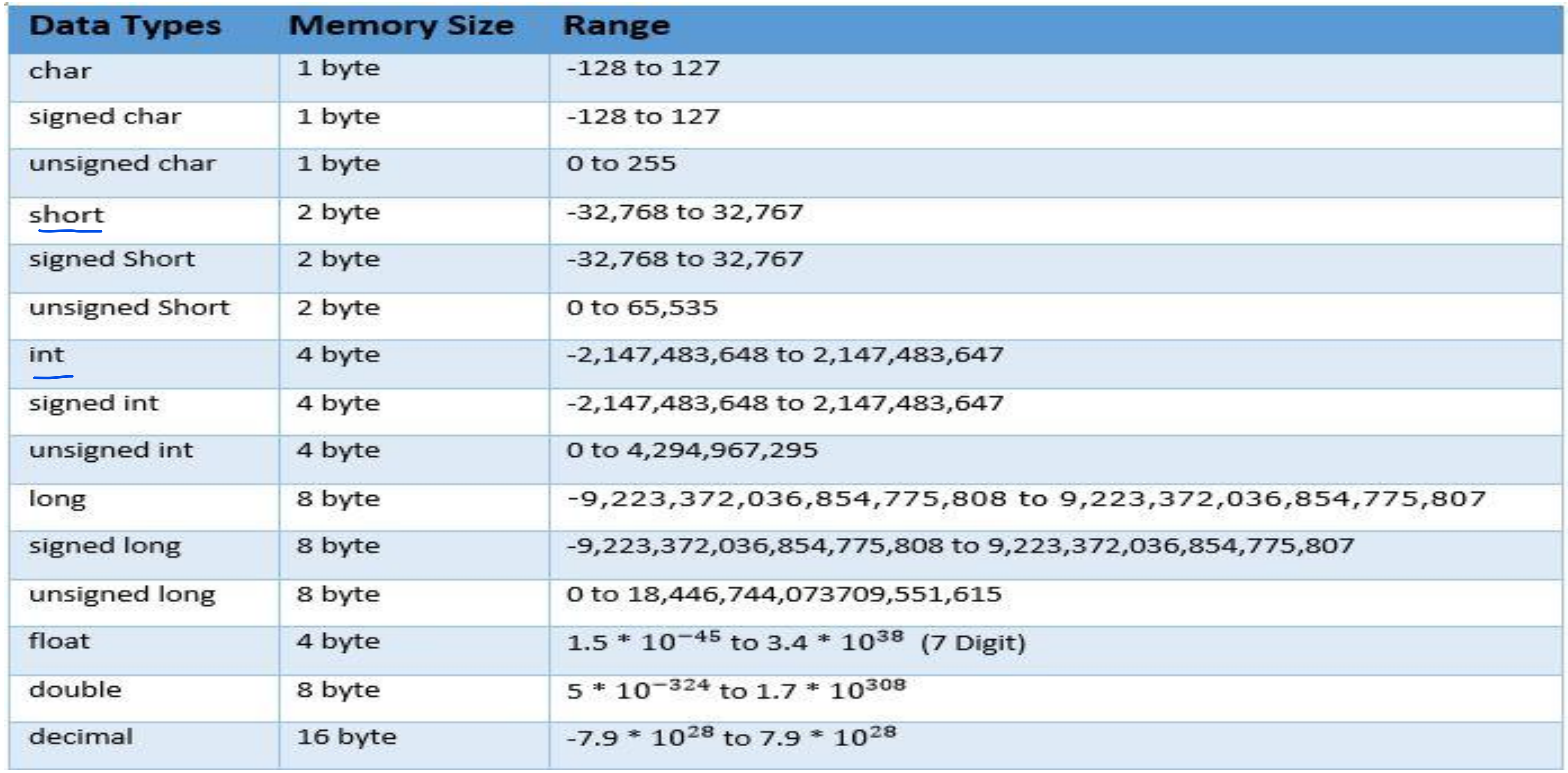
  - int ( %d %u %o %x )
  - short ( %hd, %hu )
  - long ( %ld, %lu )
  - char ( %c )
  - float ( %f )
  - double ( %lf )

- **Integer types can be signed/unsigned**

- **Derived types**
  - Array
  - Pointer
  - Function

- **User defined types**
  - struct
  - union
  - Enum

- **void type** – represent no value



| Data Types | Memory Size | Range |
|---|---|---|
| char | 1 byte | -128 to 127 |
| signed char | 1 byte | -128 to 127 |
| unsigned char | 1 byte | 0 to 255 |
| short | 2 byte | -32,768 to 32,767 |
| signed Short | 2 byte | -32,768 to 32,767 |
| unsigned Short | 2 byte | 0 to 65,535 |
| int | 4 byte | -2,147,483,648 to 2,147,483,647 |
| signed int | 4 byte | -2,147,483,648 to 2,147,483,647 |
| unsigned int | 4 byte | 0 to 4,294,967,295 |
| long | 8 byte | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| signed long | 8 byte | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| unsigned long | 8 byte | 0 to 18,446,744,073709,551,615 |
| float | 4 byte | $1.5 * 10^{-45}$ to $3.4 * 10^{38}$ (7 Digit) |
| double | 8 byte | $5 * 10^{-324}$ to $1.7 * 10^{308}$ |
| decimal | 16 byte | $-7.9 * 10^{28}$ to $7.9 * 10^{28}$ |

## Identifiers

- Helps to identify memory locations, functions, defined types and pre-processor macros.

- **Rules of Identifiers:**
  - Should start with alphabet or with _ (underscore)
  - Can include alphabets, _ (underscore),digits
  - Case sensitive

- **Examples:**
  - Var_1 //Valid
  - 1_var // Not Valid
  - _var //valid
  - Var-1 // invalid
  - Basic Salary //invalid

## Declaration, Definition, Initialisation

- **Declaration :**
  - Intimate compiler about a particular resource like memory, function, data type about its nature.
- **Definition :**
  - Here **allocates memory** for the declared resource.
- **Initialization :**
  - Applicable to memory location which can be set with specific value.
- **Assignment**
  - Giving a **value to a variable after it has been defined**.
- **Variable :**
  - A **named memory location** whose value can be changed at runtime

- **Variable examples**
  - int number = 10;
  - double basic_salary = 20000.0;
  - char letter = 'A';
  - int roll_number;
  - roll_number = 20;
  - double price = 200.0;
  - price = 300.0;
- **Each variable is assigned some memory location.**
- **Size of data type of given variable or constant is found by sizeof() operator**.

# printf()

- Arbitrary strings and variable values can be printed using printf() function.
- Use following format specifiers to format data in specific type
  - ✓ %d - to format data in signed integer
  - ✓ %u - to format data in unsigned int
  - ✓ %c - to format data in character
  - ✓ %f - to format data in float
  - %s - to format data in string
  - ✓ %ld - to format data in long integer
  - %x - to format data in hexadecimal
  - ✓ %o - to format data in octal
- **Formatting:** %5d, %-7d, %08d, %8.2f, …

**Examples:**

- printf("Hello DESD @ Sunbeam");
- printf("%d", roll_number);
- printf("%d %lf %c", number, basic_salary, letter);
- printf("Book price is %lf", price);

## scanf()

- **scanf()**
  - Used to input values from user.
  - Same format specifiers as of printf().
  - Do not use any char other than format specifiers in format string.
  - To skip a char from input use **%*c.**

e.g. **Accept float number from user**

```
float num;
scanf( “ %f “ , &num );
```

## Escape Sequence character

- Can be used with string
- Escapes the meaning of followed by character.

- **List of Escape Sequence characters available in C:** •
  - **\n** - Helps to add new line
  - **\r** - Helps to add carriage return. Moves carriage to the beginning of same line
  - **\t** - Adds horizontal tab space
  - **\b** - Moves carriage I character back
  - **\a** - Adds beep/alert

## Using Width for printing data

- **Right-align**
  - int num = 12;
  - printf(“%4d”,num); output : _ _ 12

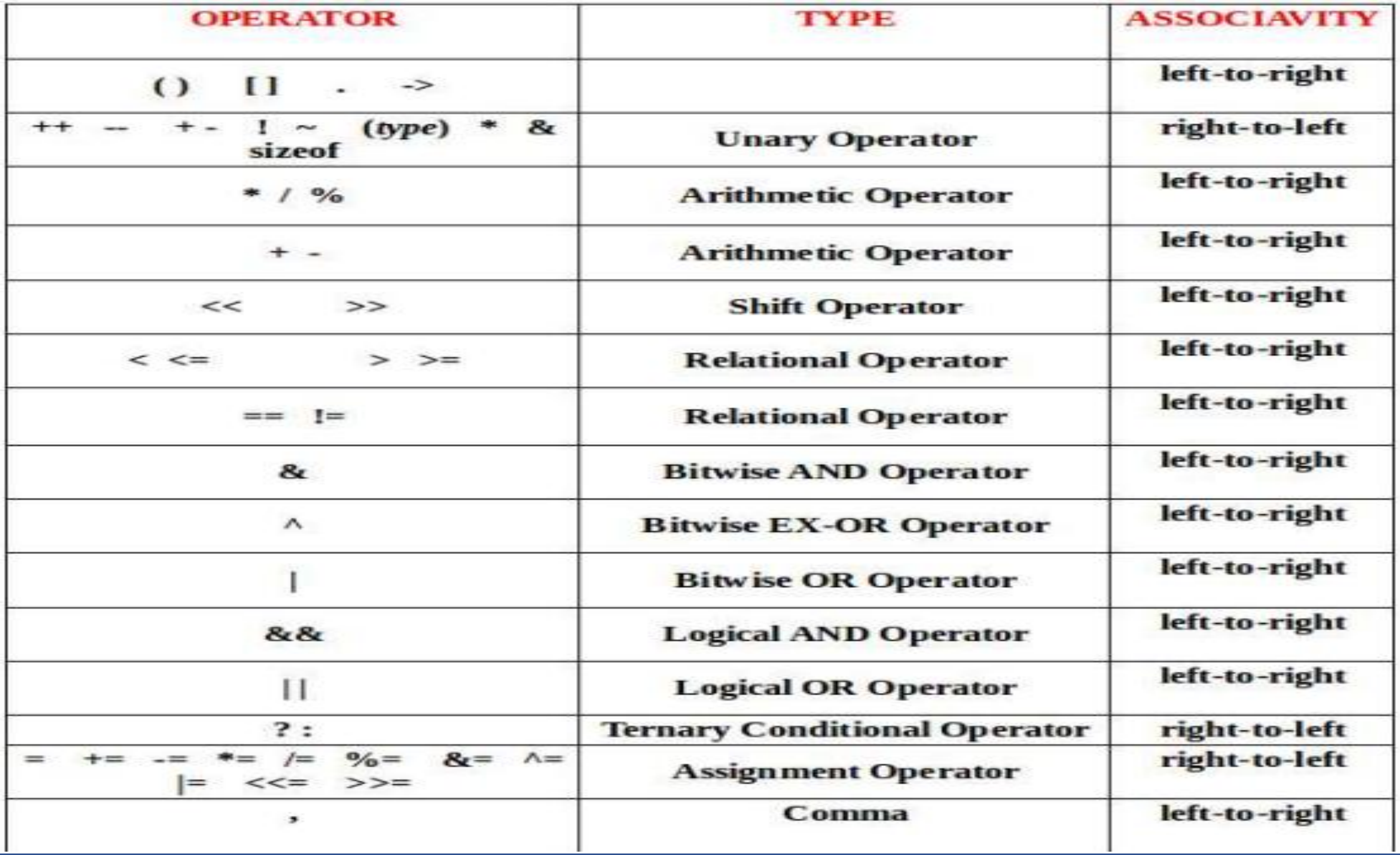
- **left-align**
  - int num = 12;
  - printf(“%-4d”,num); output : 1 2 _ _

  - float fval= 12.48;
  - printf(“%6.2f”,fval); output : _ 1 2 . 4 8

# Operators in C

|  |  |
|---|---|
| Arithmetical Operators | +  -  /  *  % |
| Logical Operators | &&  \|\|  ! |
| Relational Operators | >  <  >=  <=  ==  != |
| Bitwise Operators | &  \|  ^  <<  >>  ~ |
| Unary Operators | +  -  *  &  sizeof  ++  --  .  -> |
| Shorthand Operators | +=  -=  /=  *=  %=  &=  \|=  ^=  <<=  >>=  ~= |
| Conditional Operators (Ternary Operators) | ? : |
| Special Operators | [ ]  ( ) |
| Assignment | = |

# Operator Precedence and Associativity



| OPERATOR | TYPE | ASSOCIAVITY |
|---|---|---|
| ( )   [ ]   .   -> |  | left-to-right |
| ++   --   + -   !   ~   (*type*)   *   & sizeof | Unary Operator | right-to-left |
| * / % | Arithmetic Operator | left-to-right |
| + - | Arithmetic Operator | left-to-right |
| <<   >> | Shift Operator | left-to-right |
| < <=   > >= | Relational Operator | left-to-right |
| == != | Relational Operator | left-to-right |
| & | Bitwise AND Operator | left-to-right |
| ^ | Bitwise EX-OR Operator | left-to-right |
| \| | Bitwise OR Operator | left-to-right |
| && | Logical AND Operator | left-to-right |
| \|\| | Logical OR Operator | left-to-right |
| ? : | Ternary Conditional Operator | right-to-left |
| = += -= *= /= %= &= ^= \|= <<= >>= | Assignment Operator | right-to-left |
| , | Comma | left-to-right |

# Typecasting

- **Typecasting** (or type conversion) is the process of **changing the data type** of a variable or expression into another type.
- **Types of Conversion**
  - **Implicit Type Conversion** (Type Promotion / Type Casting by Compiler)
  - **Explicit Type Conversion** (Type Casting by Programmer)

  - **float + int → float**
  - **char + int → int**
  - **int + unsigned int → unsigned int**

# Decision Control : **If …**

- **Syntax:**

```
if (<expression>)
{
        <staements>
}// executes when expression results true
```

- Any expression which results non zero value is considered as true where as zero is considered as false.
- { } optional provided there is only one statement

## Decision Control : **If ..else**

- **Syntax:**

```
if (<expression>)
{
        <staements>
}// executes when expression results true
else
{
        <staements>
 } // executes when expression results false
```

## Decision Control : **Nested if..**

- **Syntax:**

```
if (<expression>)
{
      if (<expression>)
       {
                  <statement>
       }// executes when expression results true

}// executes when expression results true
else
{
                  <statement>
} // executes when expression results false
```

# Decision Control : **If ..else if ….**

- **Syntax:**

```
if (<expression>)                                          //1.
{
        <statement>
}// executes when expression results true
else if (<expression>)                                     //2.
{
        <statement>
 } // executes when expression 1 results false
else if (<expression>)                                      //3.
 {
        <statement>
 } // executes when expression 1,2 results false
else                                                       //4.
{
        <statement>
} // executes when expression 1,2,3 results false
```

## Conditional Operators ? :

- **Syntax:**

  **< expression >? <true> : <false> ;**

- **Points to note:**

1. Follows right to left associativity rule
2. Can not use jump statement in true or false part

## typedef

- **Syntax:**

  **typedef < existing data type > < another name/alias  > ;**

- **Points To Note:**
- 1. Helps to give another name to existing data type.
- 2. Improves readability of source code.
- 3. Helps to port code across multiple architecture/platforms.
- 4. Helps to simplify complicated declarations

## User Defined Data Type : enum

- **Syntax:**

  - **enum < tag name > {[ < enumated fields > ,...]} ;**
  - **e.g. enum colors {  RED , BLUE , GREEN };**

1. Helps to define new data type.
2. Collection of enumerated fields.
3. Each enumerated field represent integer constant and also can be initialized with integer constant.
4. Helps to improve readability of source code.
5. If we do not initialize first field by default it represents integer 0 value. And next fields carry

## Decision Control - switch case

- **Syntax :**

```
switch(<expression>)
{
     case <integer constant>>:    <statement>
                                  break;
     case <integer constant>:    <statement>
                                  break;

          …
          …
          …
     default:                          <statement>
                                 break;
}
```

## Decision Control - switch case

- **Points To Note:**

1. Each case should be followed by integer constant.
2. Can not add duplicate cases.
3. Use of break is suggested in each case at the last. Else execution control is given to next case even though next case is not satisfied.
4. Use of default case is optional.
5. Sequence of case does not matter.

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com