C Programing

Trainer : Kiran Jaybhave

C-programing

- ❖Data Types
- ❖Format Specifies
- ❖Escape Sequence Characters
- ❖Sizeof operator
- ❖Width Specifier
- ❖Type Modifiers
- ❖Typecasting
- ❖Conditional Statement : **if else**

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, 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**
  - **int -------------------------------- >**
  - **Char -------------------------------- >**
  - **float -------------------------------- >**
  - **double -------------------------------- >**
- **Integer types can be signed/unsigned**
- **Derived types**
  - Array
  - Pointer
  - Function

- printf() format specifiers

%d, %u, %o, %x , %i
%c
%f
%f ,%g

- **User defined types**
  - struct
  - union
  - enum
- **void type – represent no value**

- **Type qualifiers**
  - const and volatile

- **Type Modifier**
  - **Signed**
  - **unsigned**
  - **Short**
  - **long**

- **Typecasting**
  - conversion of data type
    - changing type of data
  - while performing arithmetic one of data type is promoted to higher data type
    - float + (float)int => float
    - (int)char + int => int
    - uint + (uint)int => uint

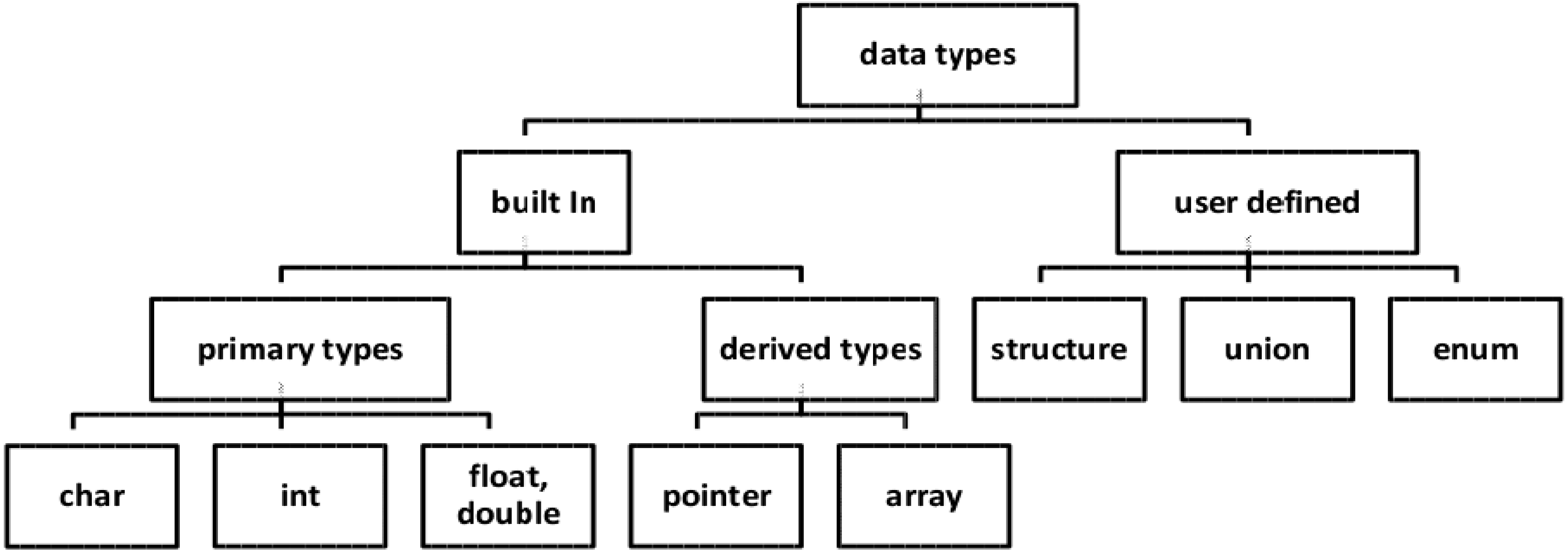
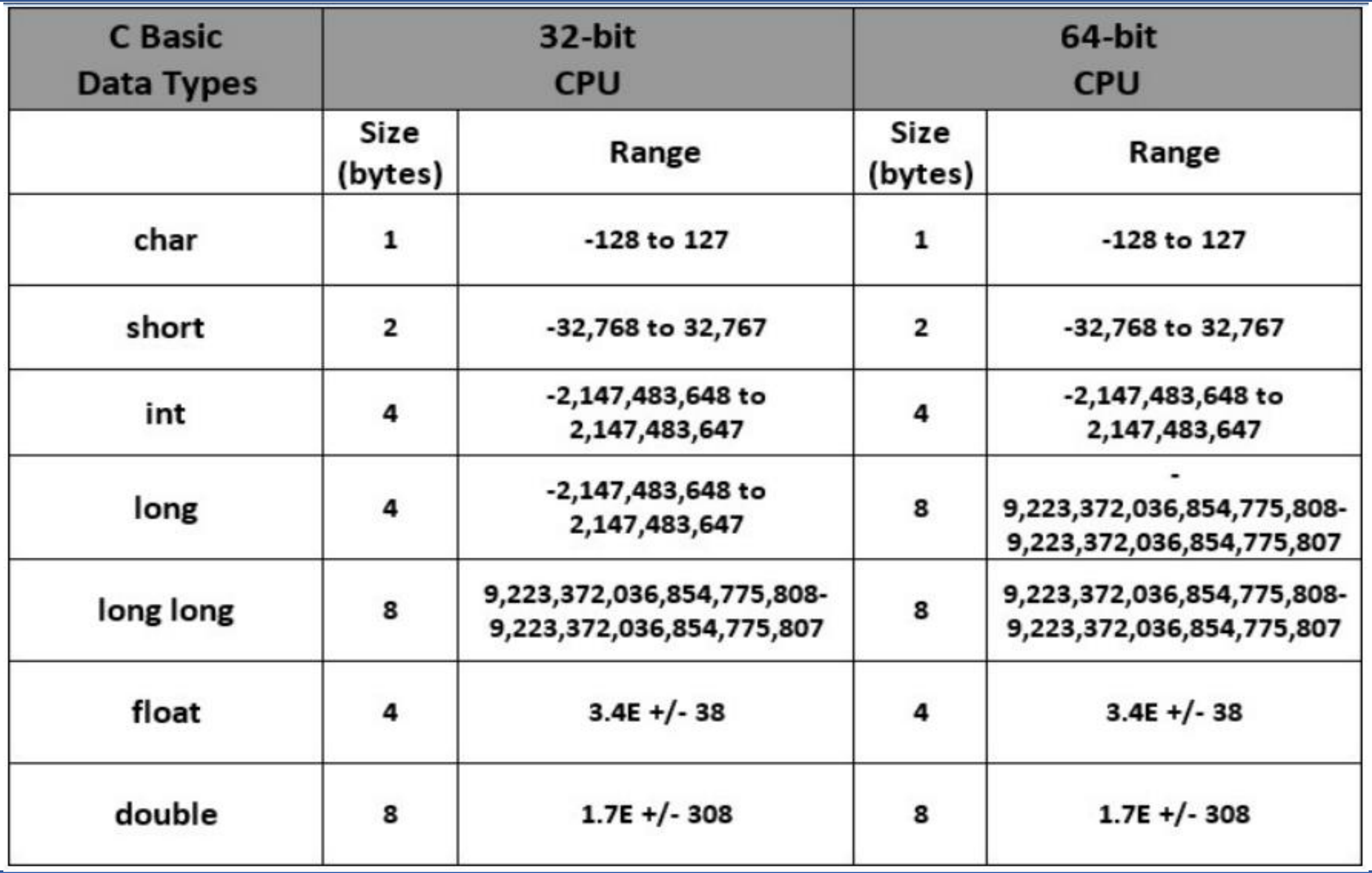


Figure 2.1 Data Types

Data Types, Variables & Constants



C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	- 9,223,372,036,854,775,808- 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

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

Examples:

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

printf() and scanf()

- #include -- function declaration
- printf()
  - Used to print values & string on terminal.
  - Various format specifiers %d, %c, %f, …
  - Formatting: %5d, %-7d, %08d, %8.2f, …
- 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.

Using Width for printing data

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

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

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

Escape Sequence character

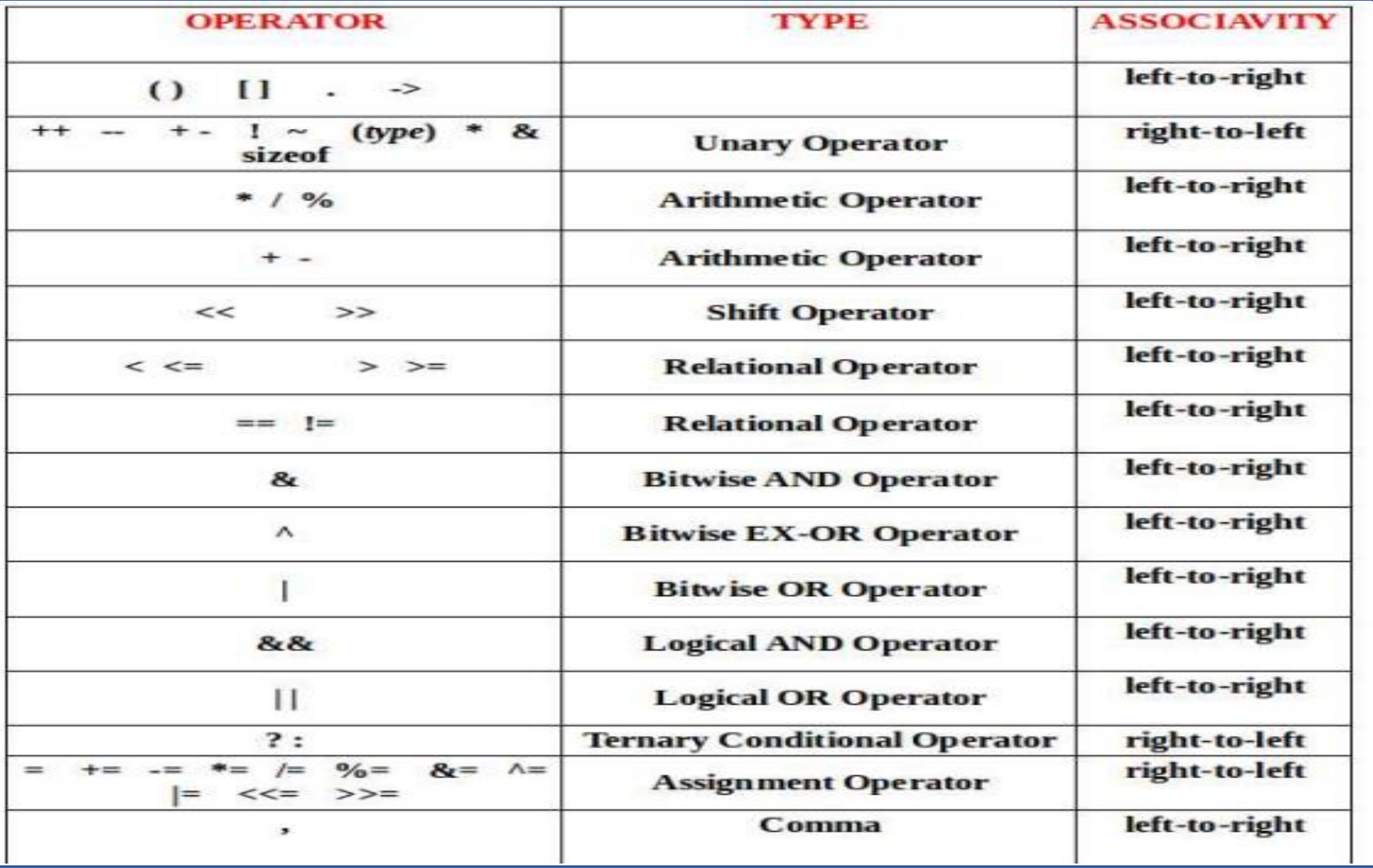
- Can be used with string
- Escapes the meaning of followed by character.

- **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
  - **\v** Adds vertical tab space. Result can be seen on printer
  - etc..

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

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.

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

SUNBEAM

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com