



C, C++, Database Course

TOPS TECHNOLOGIES

Module 1 Topics

- **Programming Language**
- **High level Programming language**
- **POP**
- **C Programming**
 - Overview, Identifiers, Constants
 - Variable, Storage classes, Symbolic Constants
 - Operators, Expressions
 - Type Conversion
 - Decision Making and branching statements
 - Array, String, Functions
 - Structure , Unions , Pointer
 - String Function, Dynamic Memory Allocation
 - Linked list , Preprocessor
 - File Management

Programming Language

- A programming language is a set of commands, instructions, and other syntax used to create software.
- Example: For writing, a meaningful story in English language we use English alphabets, words and grammar.
- Similarly, Software are created using a Programming language that has its own syntax and rules.
- Example of programming Language: Python,C, C++, Java.

High Level Language

- High level languages are written in a form that is close to our human language, enabling the programmer to just focus on the problem being solved
- No particular knowledge of the hardware is needed as high level languages create programs that are portable and not tied to a particular computer or microchip.
- These programmer friendly languages are called 'high level' as they are far removed from the machine code instructions understood by the computer.
- Examples include: C,C++, Java, Pascal, Python, Visual Basic.

Advantages

- Easier to modify as it uses English like statements
- Easier/faster to write code as it uses English like statements
- Easier to debug during development due to English like statements
- Portable code – not designed to run on just one type of machine

Low Level Language

- Low level languages are used to write programs that relate to the specific architecture and hardware of a particular type of computer.
- They are closer to the native language of a computer (binary), making them harder for programmers to understand.
- Examples of low level language:
Assembly Language, Machine Code

Procedural Programming

- POP follows a step-by-step approach to break down a task into a sequence of instructions.
- Each step is carried out in order in a systematic manner so that a computer can understand what to do. The program is divided into small parts called functions and then it follows a series of computational steps to be carried out in order.
- It follows a top-down approach to actually solve a problem, hence the name.
- Procedures correspond to functions and each function has its own purpose.
- Dividing the program into functions is the key to procedural programming. So a number of different functions are written in order to accomplish the tasks
- E.g.: c, basic, FORTRAN.

Why C?

- C is small (**only 32 keywords**).
- C is common (lots of C code about).
- C is stable (the language doesn't change much).
- C is quick running.
- C is the basis for many other languages (Java, C++, awk, Perl,C#).
- It may not feel like it but C is one of the easiest languages to learn.

Advantages of C language

- C is a general purpose programming language, meaning that it is not limited to any one specific kind of programming. You can write all sorts of software using C.
- ~~C is not a very high level language.~~ C allows you to directly access memory addresses, create bit fields and structures and map them to memory, perform bitwise operations and so on. C facilitates hardware programming.
- ~~Not being high level also means there is little overhead; it is~~ highly efficient and provides fast execution speed.
- There are C language compilers and development tools available for many different platforms from small embedded systems to large mainframes and supercomputers. .

Structure of C programming

```
#include <stdio.h> // Library Files  
void main()  
{  
    printf("Hello World"); // printing statement  
}
```

Hello World

Refer this Example:

[\(printing hello world\)](https://github.com/TopsCode/Software-Engineering/blob/master/C/helloWorld.c)

Structure of C programming

10

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

- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations.
- This is called preprocessor in C.

```
void main()
```

- The second line main() tell the compiler that it is the starting point of the program, every program should essentially have the main function only once in the program.
- The opening and closing braces indicates the beginning and ending of the program. All the statements between these two braces form the function body. These statements are actually the C code which tells the computer to do something.

printf()

- Printf functions (which stands for "print formatted") are a class of functions typically associated with some types of programming languages. They accept a string parameter called the format string
- The `/* */` is a comment and will not be executed, the compiler simply ignores this statement. These are essential since it enhances the readability and understandability. of the program

Keywords and Identifiers

Keyword:

- Keywords are predefined reserved identifiers that have special meanings
- They ~~cannot be used as identifiers in your program.~~

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

Identifier:

- Identifiers refers to the name of user-defined variables and functions.

Rules for Identifiers

The identifiers must conform to the following rules:

1. First character must be an alphabet (or underscore)
2. Identifier names must consists of only letters, digits and underscore.
3. A identifier name should have less than 31 characters.
4. Any standard C language keyword cannot be used as a variable name.
5. A identifier should not contain a space.



Variables in C

- It is a data name which is used to store data and may change during program execution.
- Variable name is a name given to memory cells location of a computer where data is stored.

Rules for variables:

- First character should be letter or alphabet.
- Keywords are not allowed to use as a variable name.
- White space is not allowed.
- C is case sensitive i.e. UPPER and lower case are significant.
- Only underscore, special symbol is allowed between two characters
- The length of identifier may be upto 31 characters but only the first 8 characters are significant by compiler.
- Some compilers allow variable names whose length may be upto 247 characters. But, it is recommended to use maximum 31 characters in variable name. Large variable name leads to

Constants in C

- A constant is an entity that doesn't change during the execution of a program.

Followings are the different types of constants :

1. Real Constant :

- It must have at least one digit.
- It must have a decimal point which may be positive or negative.
- Use of blank space and comma is not allowed between real constants.

Example:

+194.143, -416.41



2. Integer Constant :

- It must have at least one digit.
- It should not contain a decimal place.
- It can be positive or negative.
- Use of blank space and comma is not allowed between real constants.
- Example: 1990, 194, -394

3. Character Constant :

- It is a single alphabet or a digit or a special symbol enclosed in a single quote.
- Maximum length of a character constant is 1.
- Example: 'T', '9', '\$'

4. String Constant :

- It is collection of characters enclosed in double quotes.
- It may contain letters, digits, special characters.
- Example: "Technowell Web Solutions, Sangli"

Data Types in C

- “Data type can be defined as the type of data of variable or constant store.”
- When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C.
- Followings are the most commonly used data types in C

Keyword	Format Specifier	Size	Data Range
char	%c	1 Byte	-128 to +127
unsigned char	<--- >	8 Bytes	0 to 255
int	%d	2 Bytes	-32768 to +32767
long int	%ld	4 Bytes	-2³¹ to +2³¹
unsigned int	%u	2 Bytes	0 to 65535
float	%f	4 Bytes	-3.4e³⁸ to +3.4e³⁸
double	%lf	8 Bytes	-1.7e³⁰⁸ to +1.7e³⁰⁸
long double	%Lf	12-16 Bytes	-1.7e³⁰⁸ to +1.7e³⁰⁸

Qualifier :

- When qualifier is applied to the data type then it changes its size.
- Size qualifiers : short, long
- Sign qualifiers : signed, unsigned

Integer Types

- Integers are used to store whole numbers

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Float Types

Float types are used to store real numbers.

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Character Type :

Character types are used to store characters value

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

► **Void Type :**

- The void type basically means "nothing". A void type cannot hold any values. You can also declare a function's return type as void to indicate that the function does not return any value.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/dataType.c>



Enum Data Type :

This is an user defined data type having finite set of enumeration constants. The keyword 'enum' is used to create enumerated data type.

Syntax:

```
enum [data_type] {const1, const2, ...., const n};
```

Example:

```
enum mca(software, web, seo);
```

Typedef :

It is used to create new data type. But it is commonly used to change existing data type with another name.

Syntax:

```
typedef [data_type] synonym;
```

OR

```
typedef [data_type] new_data_type;
```

Example:

```
typedef int integer;  
integer rno;
```



Managing Input and Output Operations

- To display output to the user, we can use `printf()` function
- To take input from user , use `scanf()` function.

Format specifier:

Format specifier	Type of value
<code>%d</code>	Integer
<code>%f</code>	Float
<code>%lf</code>	Double
<code>%c</code>	Single character
<code>%s</code>	String
<code>%u</code>	Unsigned int
<code>%ld</code>	Long int
<code>%lf</code>	Long double

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/inputOutput1.c>

Single character input output:

The basic operation done in input output is to read a characters from the standard input device such as the keyboard and to output or writing it to the output unit usually the screen. The getchar function can be used to read a character from the standard input device. The scanf can also be used to achieve the function. The getchar has the following form.

```
character_variable = getchar();
```

- Variable name is a valid 'C' variable, that has been declared already and that possess the type char.

Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/simpleProgram.c>



The putchar function which is analogous to getchar function can be used for writing characters one at a time to the output terminal. The general form is

`putchar (variable name);`

Where variable is a valid C type variable that has already been declared Ex:-

`putchar (ch);`

Displays the value stored in variable ch to the standard screen.

Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/inputOutput.c>

Declaration of Storage Classes

25

- A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.
- There are following storage classes which can be used in a C Program
 - auto
 - register
 - static
 - Extern

auto - Storage Class

- auto is the default storage class for all local variables.
- Keyword : auto
- Storage Location : Main memory
- Initial Value : Garbage Value
- Life : Control remains in a block where it is defined.
- Scope : Local to the block in which variable is declared.
- Refer the Example:
- <https://github.com/TopsCode/Software-Engineering/blob/master/C/autoExample.c>



Register Storage Class :

Keyword : register

Storage Location : CPU Register

Initial Value : Garbage

Life : Local to the block in which variable is declared.

Scope : Local to the block.

Syntax : register [data_type] [variable_name];

Static Storage Class :

Keyword : static

Storage Location : Main memory

Initial Value : Zero and can be initialize once only.

Life : depends on function calls and the whole application or program.

Scope : Local to the block.

Syntax : static [data_type] [variable_name];

Example : static int a;

Refer this example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/register.c>

Types of Static Variable:

There are two types of static variables as :

- a) Local Static Variable
- b) Global Static Variable

Static storage class can be used only if we want the value of a variable to persist between different function calls.

Refer this example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/staticVariable.c>



External Storage Class :

Keyword : extern

Storage Location : Main memory

Initial Value : Zero

Life : Until the program ends.

Scope : Global to the program.

Syntax : extern [data_type] [variable_name];

Example : extern int a;

Refer this example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/extenVariable.c>

Defining Symbolic Constants

A symbolic constant value can be defined as a preprocessor statement and used in the program as any other constant value. The general form of a symbolic constant is

```
#define symbolic_name value_of_constant
```

Valid examples of constant definitions are :

```
# define MARKS 100  
# define TOTAL 50  
# define PI 3.14159
```

These values may appear anywhere in the program, but must come before it is referenced in the program.

It is a standard practice to place them at the beginning of the program.

Volatile Variable

- A volatile variable is the one whose values may be changed at any time by some external sources.
- Example:
`volatile int num;`
- The value of data may be altered by some external factor, even if it does not appear on the left hand side of the assignment statement.
- When we declare a variable as volatile the compiler will examine the value of the variable each time it is encountered to see if an external factor has changed the value



Operators

- "Operator is a symbol that is used to perform mathematical operations."
- When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C.
- Followings are the most commonly used data types in C.

Operator Name	Operators
Assignment	=
Arithmetic	+ , - , * , / , %
Logical	&& , , !
Relational	< , > , <= , >= , == , !=
Shorthand	+ = , - = , * = , / = , % =
Unary	++ , --
Conditional	(exp1)? exp2: exp3 ;

Assignment Operator: It is used to assign a value to variable.

Arithmetic Operator: It is also called as 'Binary operators'. It is used to perform arithmetical operations. These operators operate on two operands.

Logical Operator: They compare or evaluate logical and relational expressions.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT



- **Logical AND (&&):**

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

- **Logical OR(||):**

a	b	a b
true	true	true
true	false	true
false	true	true
false	false	false

- **Logical Not**

Boolean NOT	
0	1
1	0

Refer this Example:

- **Assignment Operator**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/AssignmentOperator.c>

- **Arithmetic Operator Example**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ArthmeticOperator.c>

- **Logical Operator Example**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/logicalOperator.c>



Relational operator

- Relational operator is used when there is relation between 2 or more variables.

Operator	Meaning
<	Less than
<=	Less than or equal to
>	greater than
>=	greater than or equal to
!=	Not equal to
==	Is equal to

```
int age=90;
if(age>18)
{
    printf("\n u are eligible for watting..")
}
```



Shorthand Operator

- It is used to perform mathematical operations at which the result or output can affect on operands.

Unary Operator

- It operates on a single operand. Therefore, this operator is called as 'unary operator.' It is used to increase or decrease the value of variable by 1.

Conditional Operator

- Conditional operator is also called as 'ternary operator.'
- It is widely used to execute condition in true part or in false part. It operates on three operands.
- The logical or relational operator can be used to check conditions.



Refer this Example:

Shorthand Operator

<https://github.com/TopsCode/Software-Engineering/blob/master/C/shortHandOperator.c>

Unary Operator:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/UnaryOperator.c>

Conditional Operator:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/conditionalOperator.c>



Special Operator:

- C supports some special operators of interest such as comma operator, size of operator, pointer operators (& and *) and member selection operators (. and ->).

The Comma Operator :

- The comma operator can be used to link related expressions together. A comma-linked list of expressions are evaluated left to right and value of right most expression is the value of the combined expression.

Some Example of comma operator:

- `(x = 10, y = 5, x + y);`
- `for (n=1, m=10, n <=m; n++,m++)`
- `t = x, x = y, y = t;`



The size of Operator :

- The operator size of gives the size of the data type or variable in terms of bytes occupied in the memory.
- The operand may be a variable, a constant or a data type qualifier.

Example

```
m = sizeof (sum);  
n = sizeof (long int);  
k = sizeof (235L);
```

- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/C/sizeOf.c>

C Programming – Expressions

❖ Arithmetic Expressions

- An expression is a combination of variables constants and operators written according to the syntax of C language.
- In C every expression evaluates to a value .
- Algebraic Expression - C Expression

a * b - c -> a * b - c
(m + n) (x + y) -> (m + n) * (x + y)
(ab / c) -> a * b / c
3x² +2x + 1 -> 3*x*x+2*x+1
(x / y) + c -> x / y + c



Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form

Variable = expression;

Example of evaluation statements are

```
x = a * b - c  
y = b / c * a  
z = a - b / c + d;
```

Refer this example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/expression1.c>



Precedence in Arithmetic Operators

- An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C.
- **High priority * / % ; Low priority + -**

Rules for evaluation of expression

1. First parenthesized sub expression left to right are evaluated.
2. If parenthesis are nested, the evaluation begins with the innermost sub expression.
3. The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
4. The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.
5. Arithmetic expressions are evaluated from left to right using the rules of precedence.
6. When Parenthesis are used, the expressions within parenthesis assume highest priority.

Refer This Example

- ▶ **Operator Precedence:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/C/Operator/precedence.c>

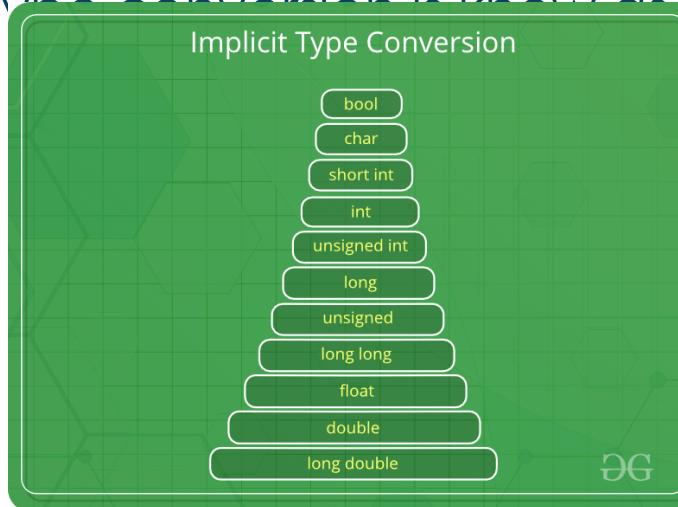


Type conversions in expressions

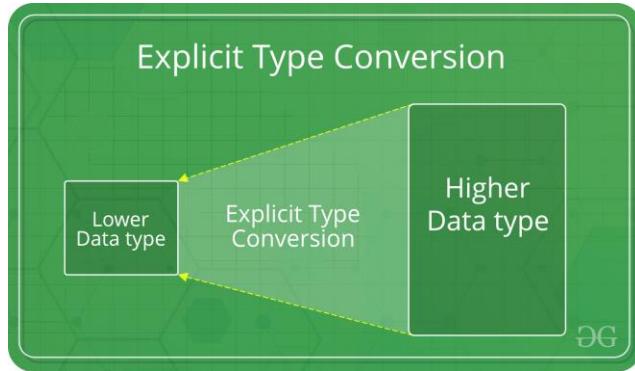
- Typecasting is also called as type conversion
- It means converting one data type into another.

❖ Implicit type conversion

- If the operands are of different types the lower type is automatically converted to the higher type before the operation proceeds. The result is of higher type.
- This automatic type conversion is known as implicit type conversion



Explicit Conversion



- Forced Casting done by programmer.
- Syntax:
`(type) expression`
- Example:
`int x=7, y=5;
float z;
z = (float)x/(float)y;`

Refer This Example

- ▶ **Implicit Conversion**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/C/typeConversion/implicit.c>
- ▶ **Explicit Conversion:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/C/typeConversion/explicit.c>

Decision Making – Branching

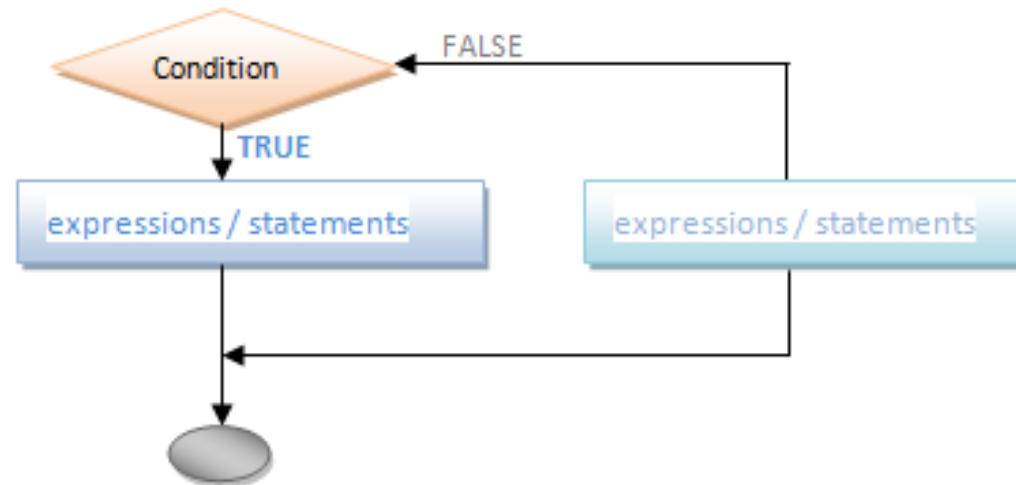
Branching :

- The C language programs presented until now follows a sequential form of execution of statements.
- Many times it is required to alter the flow of the sequence of instructions. C language provides statements that can alter the flow of a sequence of instructions.
- These statements are called control statements. These statements help to jump from one part of the program to another. The control transfer may be conditional or unconditional.



Control statements:

- ▶ 1) if statement.
- ▶ 2) if else statement.
- ▶ 3) Nested if statements.
- ▶ 4) Else if ladder statements.
- ▶ 5) Switch statement.



1. The if Statement:

- To conditionally execute statements, you can use the if or the if...else statement.
- The general form of the if statement is

```
if(expr) {  
    s1 ;  
    s2 ;  
    ....  
}
```

2. The If else construct:

- The if else is just an extension of the general format of if statement. If the result of the condition is true, then program statement 1 is executed, otherwise program statement 2 will be executed.

```
if (condition)  
    simple or compound statement  
else  
    simple or compound statement.
```



3. Nested if Statement:

- The if statement may itself contain another if statement is known as nested if statement.

Compound Relational tests:

- C language provides the mechanisms necessary to perform compound relational tests.
- A compound relational test is simple one or more simple relational tests joined together by either the logical AND(&&) or the logical OR(||) operators.

Syntax :

a> if (condition1 && condition2 && condition3)

b> if (condition1 || condition2 || condition3)



4. The ELSE If Ladder

- When a series of many conditions have to be checked we may use the ladder else if statement which takes the following general form.

```
if (condition1)
    statement – 1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
else if (condition)
    statement n;
else
    default statement;

statement-x;
```

- This construct is known as if else construct or ladder. The conditions are evaluated from the top of the ladder to downwards.

Refer this Example:

- **If statement:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/if.c>

- **If-else:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ifElse.c>

- **If else with compound Relational Test:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/compoundRelationalTest.c>

- **Nested if-else**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/NestedIfElse.c>

- **If-else Ladder:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ifElseLadder.c>

The Switch Statement:

- Switch case statements are a substitute for long if statements that compare a variable to several integral values
- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution.

```
switch(switch_expr)
{
    case constant expr1 : S1;
        S2;
        break;
    case constant expr1 : S3;
        S4;
        break;
    .....
    default      : S5;
        S6;
        break;
}
```

Refer This Example:

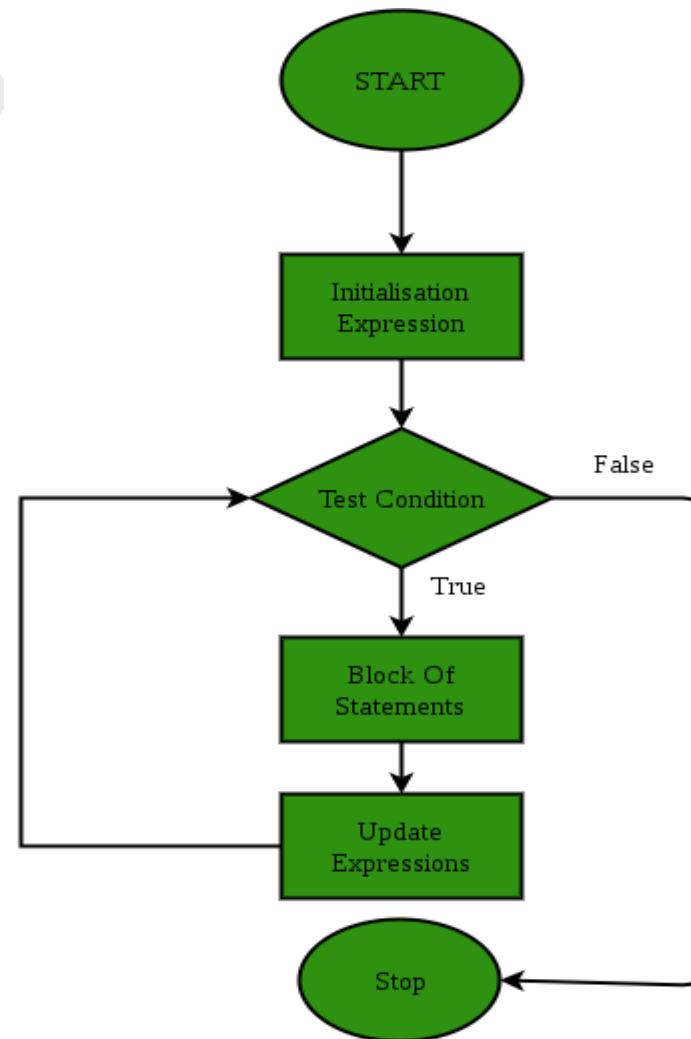
<https://github.com/TopsCode/Software-Engineering/blob/master/C/switchCase.c>

Decision Making – Looping

- During looping a set of statements are executed until some conditions for termination of the loop is encountered. A program loop therefore consists of two segments one known as body of the loop and other is the control statement. The control statement tests certain conditions and then directs the repeated execution of
 - the statements contained in the body of the loop.
 - In looping process in general would include the following four steps
 - 1. Setting and initialization of a counter
 - 2. Exertion of the statements in the loop
 - 3. Test for a specified conditions for the execution of the loop
 - 4. Incrementing the counter
 - The test may be either to determine whether the loop has repeated the specified number of times or to determine whether the particular condition has been met.

Types of loop

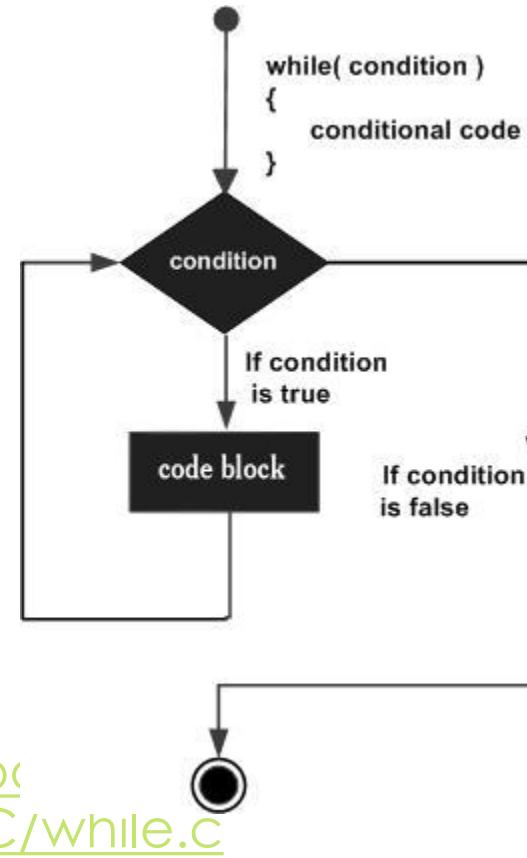
- 1) While loop,
- 2) Do.. While loop,
- 3) For loop,



The While Statement:

- A while loop in C programming repeatedly executes a target statement as long as a given condition is true

- ```
while (test condition)
{
 //body of the loop
 updatation;
}
```



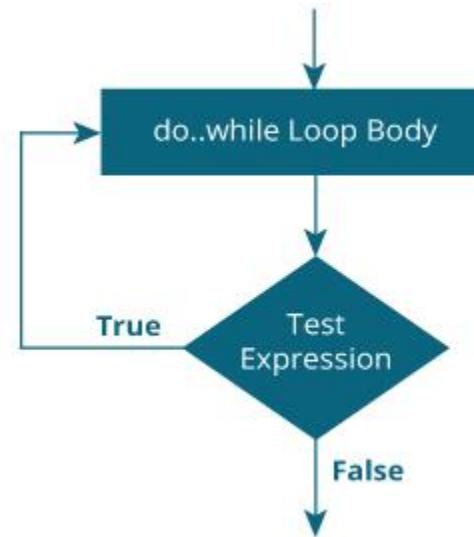
- **Refer this Example:**

<https://github.com/TopsEngineering/blob/master/C/while.c>

# The Do while statement:

- The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

```
do
{
 statement;
 updation;
}
while(condition);
```



- Refer this Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/C/DoWhile.c>

# For Loop:

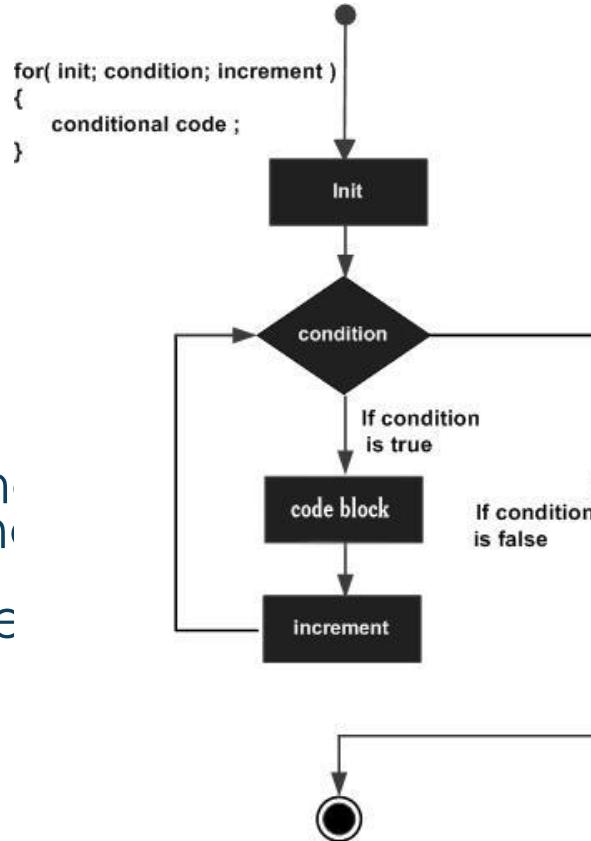
- The for loop provides a more concise loop control structure.

```
for (expr1;expr2;expr3){
 s1;
 s2 ;
}
```

The for loop is executed as follows:

- It first evaluates the initialization code.
  - Then it checks the condition expression.
  - If it is true, it executes the for-loop body.
  - Then it evaluates the increment/decrement condition and again follows from step 2.
  - When the condition expression becomes false, it exits the loop.
- Refer this example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ForLoop.c>



# Jumping Statements:

- 1) Goto Statement.
- 2) Break Statement.
- 3) Countinue Statement



## The GOTO statement:

- By using this goto statements we can transfer the control from current location to anywhere in the program.
- To do all this we have to specify a label with goto and the control will transfer to the location where the label is specified.

```
goto label;

label:


```

## Refer this Example:

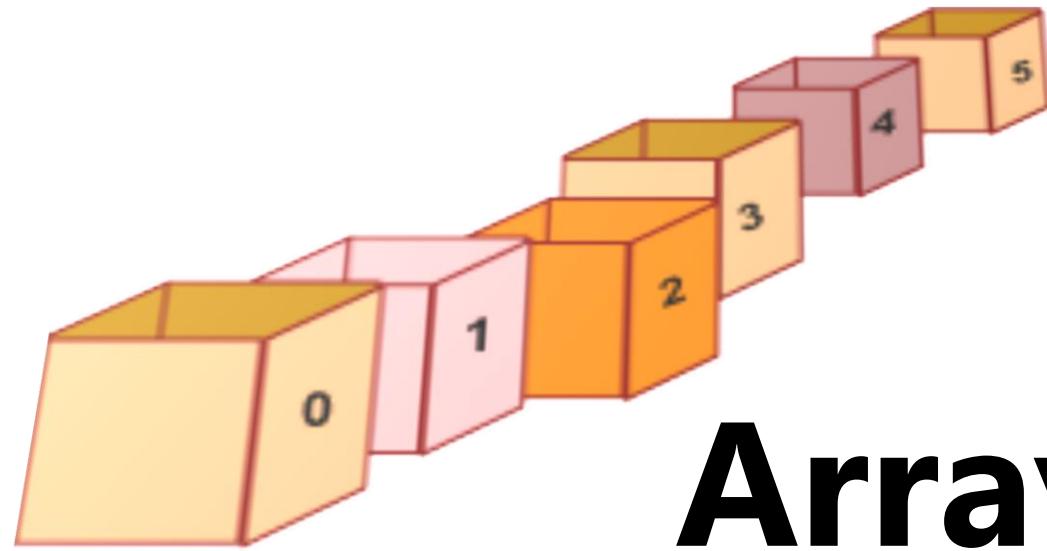
<https://github.com/TopsCode/Software-Engineering/blob/master/C/got.c>

# The Break Statement:

- The break statement is used inside loop or switch statement.
- When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.
- **Syntax:** break ;
- **Refer this Example:**  
<https://github.com/TopsCode/Software-Engineering/blob/master/C/Break.c>

## Continue statement:

- The continue statement is also used inside loop.
- When compiler finds the continue statement inside a loop, compiler will skip all the following statements in the loop and resume the next loop iteration.
- **Syntax:** continue ;
- **Refer this Example:**  
<https://github.com/TopsCode/Software-Engineering/blob/master/C/continue.c>



# Array

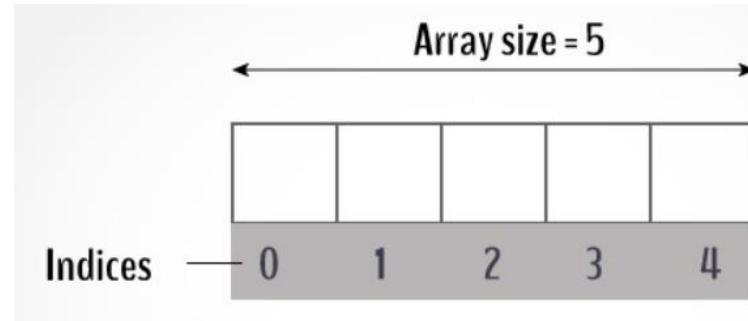


# ARRAY

- An array is a variable that can store multiple values of similar data type

- **Declaration of arrays:**

```
datatype array-name[array_size];
```



- Example:

```
float mark[5];
```

## Access Array Elements:

- Array's element can be accessed by indices.
- Arrays have

mark[0] mark[1] mark[2] mark[3] mark[4]



## Initialization of arrays:

- **Initialize during declaration:**

- Syntax:

type array\_name[size]={list of values};

- **Example:**

int mark[5] = {19, 10, 8, 17, 9};

Or

int mark[] = {19, 10, 8, 17, 9};

## Refer this Example:

- <https://github.com/TopsCode/Software-Engineering/blob/master/C/Array/example1.c>
- <https://github.com/TopsCode/Software-Engineering/blob/master/C/Array/Example2.c>

## Multi dimensional Arrays:

- In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays
- C allows arrays of three or more dimensions. The compiler determines the maximum number of dimension.
- The general form of a multidimensional array declaration is:  
`data_type array_name[s1][s2][s3]....[sn];`

`int survey[3][5][12]; //3-D array to hold 180 elements`

`float table[5][4][5][3];//4-D array containing 300 elements`

- The declaration of **two dimension arrays** is as follows:

`data_type array_name[row_size][column_size];`  
`float x[3][4];`

- The declaration of **three dimension array**:

`float y[2][4][3];`

|       | Column 1 | Column 2 | Column 3 | Column 4 |
|-------|----------|----------|----------|----------|
| Row 1 | x[0][0]  | x[0][1]  | x[0][2]  | x[0][3]  |
| Row 2 | x[1][0]  | x[1][1]  | x[1][2]  | x[1][3]  |
| Row 3 | x[2][0]  | x[2][1]  | x[2][2]  | x[2][3]  |



## Initialization of multidimensional arrays:

- Like one dimension arrays, 2 dimension arrays may be initialized by following their declaration with list of initial values enclosed in braces

Example:

```
int table[2][3]={0,0,1,1,1};
```

- Initializes the elements of first row to zero and second row to 1. The initialization is done row by row. The above statement can be equivalently written as

```
int table[2][3]={{0,0,0},{1,1,1}}
```

### Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/MultiDimensionalArray.c>



# String



# What is String?

In C programming, a string is a sequence of characters terminated with a null character \0.

For example:

```
int c[] = "c string";
```

|   |  |   |   |   |   |   |   |    |
|---|--|---|---|---|---|---|---|----|
| c |  | s | t | r | i | n | g | \0 |
|---|--|---|---|---|---|---|---|----|



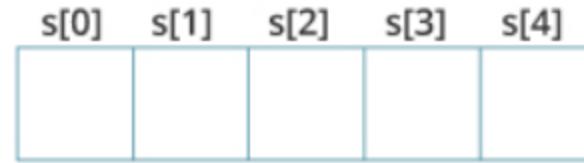
## Declare a string:

### Syntax:

```
char str_name[size];
```

### Example:

```
char s[5];
```



## Initialize a string:

```
char c[] = "abcd";
```

```
char c[50] = "abcd";
```

```
char c[] = {'a', 'b', 'c', 'd', '\0'};
```

```
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```



# Reading Strings from the terminal:

- The function `scanf` with `%s` format specification is needed to read the character string from the terminal. Example is as follow.

```
char address[15];
scanf("%s",address);
```

- The function `getchar` can be used repeatedly to read a sequence of successive single characters and store it in the array.

## Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/String/scanf.c>

<https://github.com/TopsCode/Software-Engineering/blob/master/C/StringLength.c>



# Multiline string:

- We can take input and give output of a string that consists of more than one word by using `gets` and `puts`, where `gets` is used to take input of a string from user and `puts` is used to display the string.

## Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/gets.c>

## Strings always end with the NULL character

<https://github.com/TopsCode/Software-Engineering/blob/master/C/nulCharacter.c>



# String Function Library



# String operations (string.h)

- C library supports a large number of string handling functions that can be used to array out many o f the string manipulations such as:

strlen():number of characters in the string.

strcat(): adding two are more strings

strcmp(): two strings : compare two strings.

strcpy(): copies one string over another

strlwr(): convert string to lower case

strupr(): convert string to upper case

strrev():reverse a string

- To do all the operations described here it is essential to include string.h library header file in the program.



## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/strlen.c>

<https://github.com/TopsCode/Software-Engineering/blob/master/C/stringFunctin.c>



# Functions

```
#include<stdio.h>
#include<conio.h>
void add(int x,int y)
{
 int result;
 result = x+y;
 printf("Sum of %d and %d is %d.\n",x,y,result);
}

void main()
{
 clrscr();
 add(10,15);
 add(55,64);
 add(168,325);
 getch();
}
```

Used Defined Function

Function Called



# What is a Function?

- A function is a set of statements that take inputs, do some specific computation and produces output.
- The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

## Types of functions :

1. Built in Functions
2. User Defined Functions



# 1. Built in Functions :

- These functions are also called as 'library functions'. These functions are provided by system. These functions are stored in library files.

e.g.

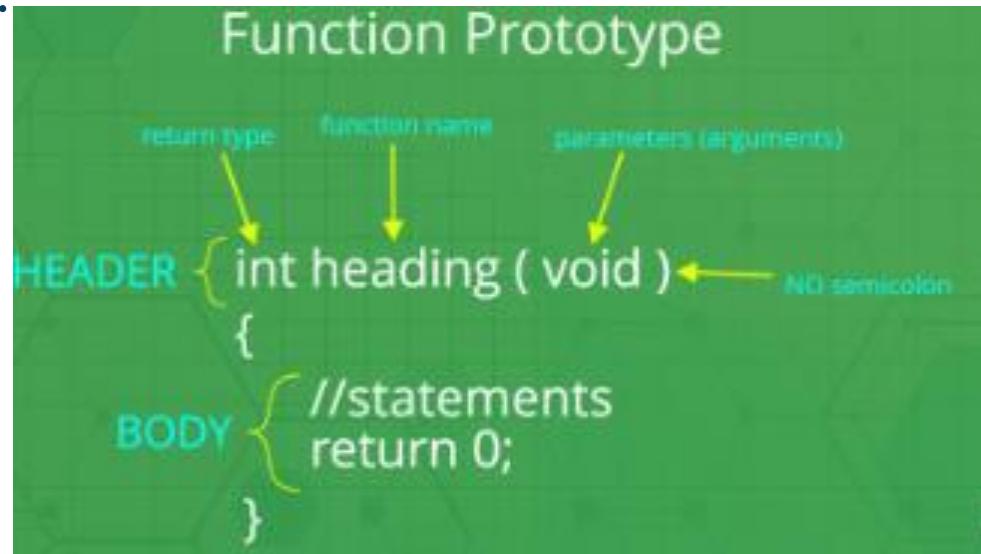
scanf()  
printf()  
strcpy()  
strlwr()  
strcmp()  
strlen()  
strcat()



## 2. User Defined Functions :

- The functions which are created by user for program are known as 'User defined functions'.

Syntax:



# Categories of user define functions

- 1) Without return type without parameter(argument).
- 2) Without return type with parameter(argument).
- 3) With return type without parameter(argument)
- 4) With return type with parameter(argument)



# Refer this example

- ▶ **Without return type without argument**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Function1.c>



# Function Call by Passing Value :

82

- When a function is called by passing value of variables then that function is known as 'function call by passing values.' Here syntax of without return type with parameter.
- Syntax:

```
// Declaration void
 return type function_name(var_nm);
 function_name(var_nm); //call

// Definition void
 return type function_name(data_type var_nm)
{
 <function_body>-----;
}
```

- Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Function2.c>



# Function Call by Returning Value :

- When a function is called by passing value of variables then that function is known as 'function call by passing values.' Here syntax of with return type with parameter.
- Syntax:

```
// Declaration
return type function_name(var_nm);

variable name = function_name(var_nm); //call

// Definition with return datatype

return type function_name(data_type var_nm)
{
 function_body; -----;
}
```

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/function3.c>

# Function Call by Passing and Returning Value :

- When a function passes and returns value of variables then that function is known as 'function call by passing and returning values.'

**Refer this example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/function4.c>



# Recursion

- Recursive function is a function that contains a call to itself. C supports creating recursive function with ease and efficient.

**Refer this example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/recursion.c>



# Structures and Unions



# What is Structures?

- A structure is a user defined data type . A structure creates a data type that can be used to group items of possibly different types into a single type.

```
struct structure_name
{
 datatype vari1, vari2...;
 datatype vari3,...;
};

struct object
{
 char id[20];
 int xpos;
 int ypos;
};
```

- Structures can group data of different types as you can see in the example of a game object for a video game. The variables you declare inside the structure are called data members.
- **Initializing a Structure**

```
struct object player1 = {"player1", 0, 0};
```



- The above declaration will create a struct object called player1 with an id equal to “player1”, xpos equal to 0, and ypos equal to 0.
- To access the members of a structure, you use the “.” (scope resolution) operator.

Sample Code:

```
struct object player1;
player1.id = "player1";
player1.xpos = 0;
player1.ypos = 0;
```

**Refer this example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/structure.c>



# Arrays of structure:

- It is possible to define a array of structures for example if we are maintaining information of all the students in the college and if 100 students are studying in the college. We need to use an array than single variables. We can define an array of structures as shown in the following example:

```
structure information
{
 int id_no;
 char name[20];
 char address[20];
 char combination[3];
 int age;
} student[100];
```

- An array of structures can be assigned initial values just as any other array can. Remember that each element is a structure that must be assigned corresponding initial values as illustrated below.
- Refer this Example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/arraysOfStructure.c>

## Structure within a structure:

- A structure may be defined as a member of another structure. In such structures the declaration of the embedded structure must appear before the declarations of other structures.

```
struct date
{
 int day;
 int month;
 int year;
};

struct student
{ int id_no;
 char name[20];
 char address[20];
 char combination[3];
 int age;
 structure date def;
 structure date doa;
}oldstudent, newstudent;
```

- stucture student contains another structure date as its one of its members.



# Union:

- Like Structures, union is a user defined data type. In union, all members share the same memory location.

```
union item
{
 int m;
 float p;
 char c;
} code;
```

- This declares a variable code of type union item. The union contains three members each with a different data type. However we can use only one of them at a time. This is because if only one location is allocated for union variable irrespective of size.



# DIFFERENCE BETWEEN STRUCTURE AND UNION

- All the members of the structure can be accessed at once, whereas in an union only one member can be used at a time. Another important difference is in the size allocated to a structure and an union.

for eg:

```
struct example
{
 int integer;
 float floating_numbers;
};
```

- The size allocated here is sizeof(int)+sizeof(float); whereas in an union union example { int integer; float floating\_numbers; } allocated is the size of the highest member. so size

# Pass Structure to a Function

- We can pass structures as arguments to functions. Unlike array names however, which always point to the start of the array, structure names are not pointers. As a result, when we change structure parameter inside a function, we don't effect its corresponding argument.

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/strucToFunc.c>

## Passing entire function to functions:

- In case of structures having to having numerous structure elements passing these individual elements would be a tedious task. In such cases we may pass whole structure to a function as shown below:



## Refer this Example:

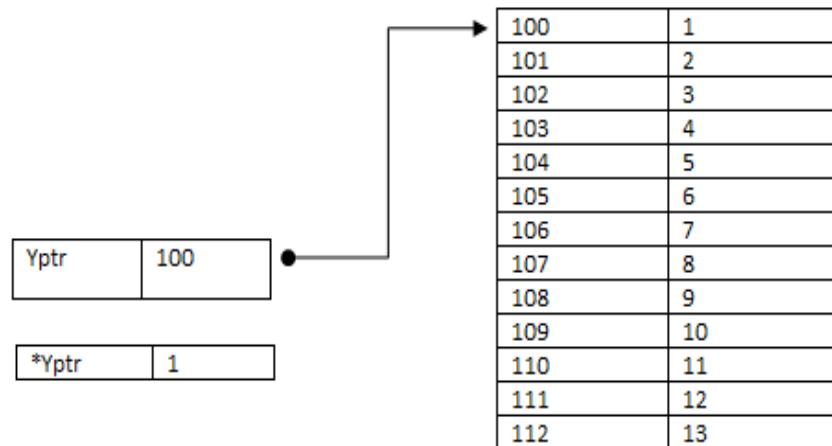
<https://github.com/TopsCode/Software-Engineering/blob/master/C/functTofunct.c>

# Pointer In C



# Introduction

- Pointers store address of variables or a memory location.
- As you can see below; Yptr is pointing to memory address 100.



# Pointer and memory relationship

## Pointer Declaration

- Declaring pointers can be very confusing and difficult at times (working with structures and pointer to pointers).
- To declare pointer variable we need to use \* operator (indirection/dereferencing operator) before the variable identifier and after data type. Pointer can only point to variable of the same data type.

## Syntax

Datatype \*pointer\_name ;

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/characterPointer.c>



# Pointer Operators

| Operator | Meaning                                                                                                                                                 |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| *        | Serves 2 purpose<br><ol style="list-style-type: none"><li>1. Declaration of a pointer</li><li>2. Returns the value of the referenced variable</li></ol> |
| &        | Serves only 1 purpose <ul style="list-style-type: none"><li>• Returns the address of a variable</li></ul>                                               |



## Address operator

- Address operator (&) is used to get the address of the operand. For example if variable x is stored at location 100 of memory; &x will return 100.
- This operator is used to assign value to the pointer variable. It is important to remember that you MUST NOT use this operator for arrays, no matter what data type the array is holding. This is because array name is pointer variable itself. When we call for ArrayA[2]; 'ArrayA' returns the address of first item in that array so ArrayA[2] is the same as saying ArrayA+=2; and will return the third item in that array.

## Pointer arithmetic

- Pointers can be added and subtracted. However pointer arithmetic is quite meaningless unless performed on arrays. Addition and subtraction are mainly for moving forward and backward in an array.
- Note: you have to be very careful NOT to exceed the array elements when you use arithmetic; otherwise you will get horrible errors such as "access violation"

| Operator             | Result                                                                |
|----------------------|-----------------------------------------------------------------------|
| <code>++</code>      | Goes to the next memory location that the pointer is pointing to.     |
| <code>--</code>      | Goes to the previous memory location that the pointer is pointing to. |
| <code>-= or -</code> | Subtracts value from pointer.                                         |
| <code>+= or +</code> | Adding to the pointer                                                 |

### Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/pointerOperator.c>

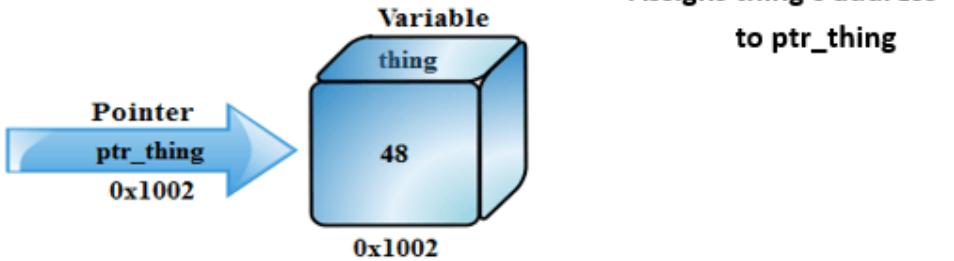
### Stack Using Pointer:

### Refer this Example:

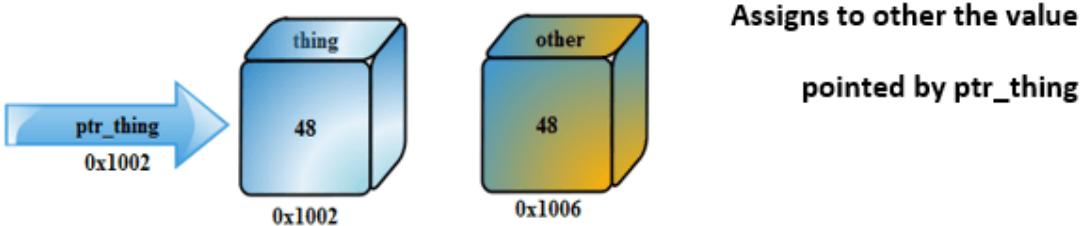
<https://github.com/TopsCode/Software-Engineering/blob/master/C/stackPointer.c>



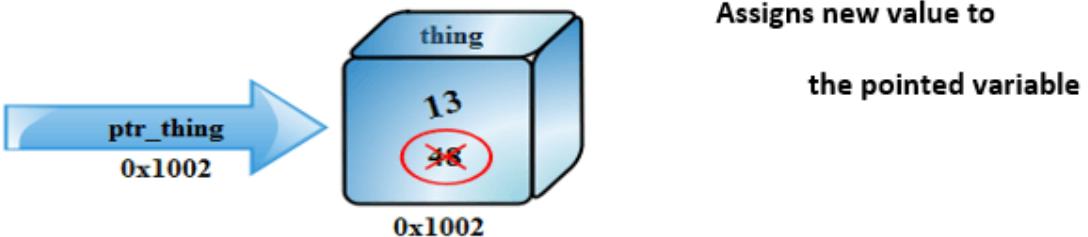
A) `ptr_thing = &thing;`



B) `other = *ptr_thing;`



C) `*ptr_thing=13;`



# Pointers and functions

- Pointers can be used with functions. The main use of pointers is ‘call by reference’ functions.
- Call by reference function is a type of function that has pointer/s (reference) as parameters to that function. All calculation of that function will be directly performed on referred variables.

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/pointerFunction.c>



- The arguments passed to function can be of two types namely
  - 1. Values passed
  - 2. Address passed
- The first type refers to call by value and the second type refers to call by reference.

### **Refer this Example:**

#### **1). Pass By Value:**

[https://github.com/TopsCode/Software-Engineering/blob/master/C/functions/pass\\_by\\_value.c](https://github.com/TopsCode/Software-Engineering/blob/master/C/functions/pass_by_value.c)

#### **2). Pass By Reference:**

[https://github.com/TopsCode/Software-Engineering/blob/master/C/functions/pass\\_by\\_refen](https://github.com/TopsCode/Software-Engineering/blob/master/C/functions/pass_by_refen)



# Pointer to arrays

- An array is actually very much like pointer. We can declare the arrays first element as `a[0]` or as `int *a` because `a[0]` is an address and `*a` is also an address the form of declaration is equivalent. The difference is pointer is a variable and can appear on the left of the assignment operator that is lvalue. The array name is constant and cannot appear as the left side of assignment operator.

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Pointers/pointerArray.c>

# Pointers and Structures

- However pointers and structures are great combinations. linked lists, stacks, queues and etc are all developed using pointers and structures in advanced systems.

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Pointers/pointerStruc.c>



# Dynamic Memory Allocation



# Dynamic memory allocation:

- Dynamic Memory Allocation can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.
- C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate dynamic memory allocation in C programming. They are:

**malloc** - Allocates memory requests size of bytes and returns a pointer to the 1st byte of allocated space

**calloc** - Allocates space for an array of elements initializes them to zero and returns a pointer to the memory

**free** - Frees previously allocated space



**realloc** - Modifies the size of previously allocated space

# Malloc:

- "malloc" or "memory allocation" method is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form.

```
ptr=(cast-type*)malloc(byte-size);
```

Example:

```
x=(int*)malloc(100*sizeof(int));
```

- Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory

**Refer This Example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Dynamic%20MDynamicAllocationmalloc.c>



## Calloc:

- “calloc” or “contiguous allocation” method is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value ‘0’.  
`ptr=(cast-type*) calloc(n,elem-size);`  
`ptr = (float*) calloc(25, sizeof(float));`
- This statement allocates contiguous space in memory for 25 elements each with the size of float.

## Free:

“free” method is used to dynamically de-allocate the memory.

```
free(ptr)
```



## realloc():

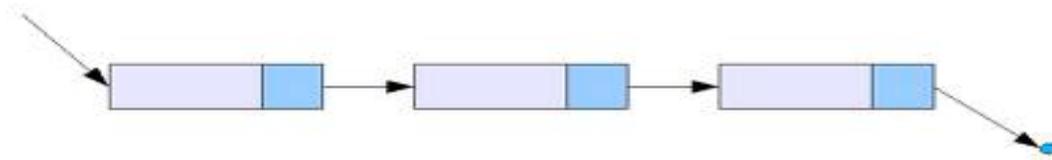
- “realloc” or “re-allocation” method is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory.
- The general statement of reallocation of memory is :  
`ptr=realloc(ptr,newsize);`

### Refer this Example:

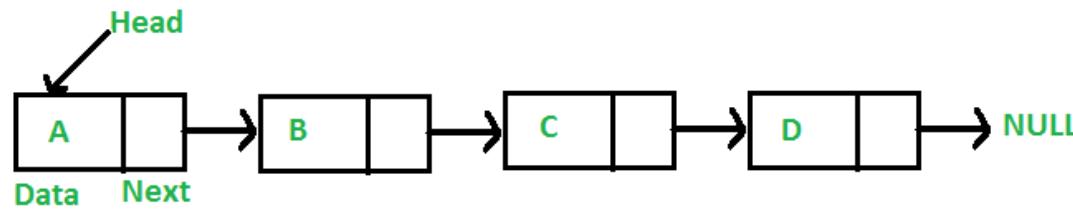
<https://github.com/TopsCode/Software-Engineering/blob/master/C/Dynamic%20Memory%20Allocation/realloc.c>



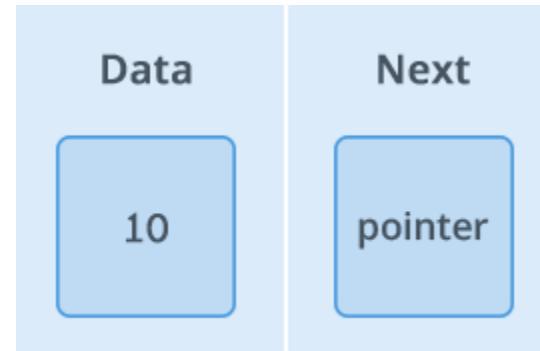
# Linked Lists



- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
- The elements in a linked list are linked using pointers as shown in the below diagram.



- a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.



- Consider the following example to illustrate the concept of linking. Suppose we define a structure as follows

```
struct linked_list
{
 float age;
 struct linked_list *next;
}
```

```
struct Linked_list node1,node2;
```

- this statement creates space for nodes each containing 2 empty fields



- The next pointer of node1 can be made to point to the node 2 by the same statement.  
`node1.next=&node2;`
- This statement stores the address of node 2 into the field node1.next and this establishes a link between node1 and node2 similarly we can combine the process to create a special pointer value called null that can be stored in the next field of the last node

# Advantages of Linked List:

- A linked list is a dynamic data structure and therefore the size of the linked list can grow or shrink in size during execution of the program. A linked list does not require any extra space therefore it does not waste extra memory. It provides flexibility in rearranging the items efficiently.
- The limitation of linked list is that it consumes extra space when compared to a array since each node must also contain the address of the next item in the list to search for a single item in a linked list is cumbersome and time consuming.



## Types of linked list:

There are different kinds of linked lists they are

1. Linear singly linked list
2. Circular singly linked list
3. Two way or doubly linked list
4. Circular doubly linked list.



# Refer this Example:

## **Linear Linked List:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Linked%20List/Linear.c>



# File management in C



- C supports a number of functions that have the ability to perform basic file operations, which include:
  1. Naming a file
  2. Opening a file
  3. Reading from a file
  4. Writing data into a file
  5. Closing a file
- Real life situations involve large volume of data and in such cases, the console oriented I/O operations pose two major problems .
- It becomes cumbersome and time consuming to handle large volumes of data through terminals.
- The entire data is lost when either the program is terminated or computer is turned off therefore it is necessary to have more flexible approach where data can be stored on the disks and read whenever necessary, without destroying the c

## File operation functions in C:

`fopen()` - Creates a new file for use Opens a new existing file for use

`fclose()` - Closes a file which has been opened for use

`getc()` - Reads a character from a file

`putc()` - Writes a character to a file

`fprintf()` - Writes a set of data values to a file

`fscanf()` - Reads a set of data values from a file

`getw()` - Reads a integer from a file

`putw()` - Writes an integer to the file

`fseek()` - Sets the position to a desired point in the file

`ftell()` - Gives the current position in the file

`rewind()` - Sets the position to the begining of ...



## Defining and opening a file:

- If we want to store data in a file into the secondary memory, we must specify certain things about the file to the operating system. They include the filename, data structure, purpose.
- The general format of the function used for opening a file is

```
FILE *fp;
fp=fopen("filename","mode");
```



# File Mode

- ▶ **r** – Opens a file in read mode and sets pointer to the first character in the file
- ▶ **w** – Opens a file in write mode. It returns null if file could not be opened. If file exists, data are overwritten.
- ▶ **a** – Opens a file in append mode. It returns null if file couldn't be opened.
- ▶ **r+** – Opens a file for read and write mode and sets pointer to the first character in the file.
- ▶ **w+** – opens a file for read and write mode and sets pointer to the first character in the file.
- ▶ **a+** – Opens a file for read and write mode and sets pointer to the first character in the file. But, it can't modify existing contents.



# Closing a file:

- The input output library supports the function to close a file; it is in the following format.

```
fclose(file_pointer);
```

- A file must be closed as soon as all operations on it have been completed. This would close the file associated with the file pointer.
- Observe the following program.

```
FILE *p1 *p2;
p1=fopen ("Input","w");
p2=fopen ("Output","r");
....
...
fclose(p1);
fclose(p2)
```

- The above program opens two files and closes them after all operations on them are completed, once a file is closed its file pointer can be reversed on other file.

- The **getc** and **putc** functions are analogous to getchar and putchar functions and handle one character at a time.
- The putc function writes the character contained in character variable c to the file associated with the pointer fp1. ex  
putc(c,fp1); similarly getc function is used to read a character from a file that has been open in read mode.  
`c=getc(fp2).`
- The program shown below displays use of a file operations. The data enter through the keyboard and the program writes it. Character by character, to the file input. The end of the data is indicated by entering an EOF character, which is control+z. the file input is closed at this signal.

### Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/File%20Management/file1.c>



## The getw and putw functions:

These are integer-oriented functions. They are similar to get c and putc functions and are used to read and write integer values. These functions would be useful when we deal with only integer data. The general forms of getw and putw are:

```
putw(integer,fp);
getw(fp);
```

### Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/File%20Management/getPut.c>



## The fprintf & fscanf functions:

- The fprintf and scanf functions are identical to printf and scanf functions except that they work on files. The first argument of these functions is a file pointer which specifies the file to be used. The general form of fprintf is `fprintf(fp,"control string", list);`
- Where fp is a file pointer associated with a file that has been opened for writing. The control string is file output specifications list may include variable, constant and string.

```
fprintf(f1,"%s%d%f",name,age,7.5);
```

- Here name is an array variable of type char and age is an int variable
- The general format of fscanf is  
`fscanf(fp,"controlstring",list);`

This statement would cause the reading of items in the  
`fscanf(f2,"5s%d",item,&quantity");`



- Like scanf, fscanf also returns the number of items that are successfully read.

# Refer This Example

- ▶ **Program to handle Mixed Data Type:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/C/File%20Management/mixedDataType.c>



## Random access to files:

- Sometimes it is required to access only a particular part of the file and not the complete file. This can be accomplished by using the following function:

### Fseek() function:

- The general format of fseek function is as follows:  
`fseek(file pointer, offset, position);`
- This function is used to move the file position to a desired location within the file. Fileptr is a pointer to the file concerned. Offset is a number or variable of type long, and position is an integer number. Offset specifies the number of positions (bytes) to be moved from the location specified by the position. The position can take the 3 values.

| Value | Meaning               |
|-------|-----------------------|
| 0     | Begnning of the File. |
| 1     | Current Position.     |
| 2     | End of the File.      |

# The Preprocessor



# The Preprocessor

- A unique feature of c language is the preprocessor. A program can use the tools provided by preprocessor to make his program easy to read, modify, portable and more efficient.
- Preprocessor is a program that processes the code before it passes through the compiler. It operates under the control of preprocessor command lines and directives. Preprocessor directives are placed in the source program before the main line before the source code passes through the compiler it is examined by the preprocessor for any preprocessor directives. If there is any appropriate actions are taken then the source program is handed over to the compiler.
- Preprocessor directives follow the special syntax rules and begin with the symbol `#` in column1 and do not require any semicolon at the end.

# Preprocessor directives:

- There are following types of preprocessor directories are availables.
- Directive – Function

| Directive       | Function                                    |
|-----------------|---------------------------------------------|
| <b>#define</b>  | Defines a macro substitution                |
| <b>#undef</b>   | Undefine a macro                            |
| <b>#include</b> | Specifies a file to be included             |
| <b>#ifdef</b>   | Tests for macro definition                  |
| <b>#endif</b>   | Specifies the end of #if                    |
| <b>#ifndef</b>  | Tests whether the macro is not def          |
| <b>#if</b>      | Tests a compile time condition              |
| <b>#else</b>    | Specifies alternatives when # if test fails |

- The preprocessor directives can be divided into three categories.
  1. Macro substitution division
  2. File inclusion division
  3. Compiler control division

## Macros:

- Macro substitution is a process where an identifier in a program is replaced by a pre defined string composed of one or more tokens we can use the #define statement for the task.
- It has the following form  
`#define identifier_string`
- The preprocessor replaces every occurrence of the identifier int the source code by a string. The definition should start with the keyword #define and should follow on identifier and a string with at least one blank space between them. The string may be any text and identifier must be a valid c name.
- There are different forms of macro substitution. The most common form is...
  1. Simple macro substitution
  2. Argument macro substitution
  3. Nested macro substitution

## Simple macro substitution:

- Simple string replacement is commonly used to define constants example:

```
#define PI 3.1415926
```

- Writing macro definition in capitals is a convention not a rule a macro definition can include more than a simple constant value it can include expressions as well. Following are valid examples:

```
#define AREA 12.36
```



# Macros as arguments:

- The preprocessor permits us to define more complex and more useful form of replacements it takes the following form.

```
#define identifier(f1,f2,f3....fn) string
```

- There is no space between identifier and left parentheses and the identifier f1,f2,f3 .... Fn is analogous to formal arguments in a function definition.
- There is a basic difference between simple replacement discussed above and replacement of macro arguments is known as a macro call
- A simple example of a macro with arguments is...

```
#define CUBE (x) (x*x*x)
```

- If the following statements appears later in the program,  
`volume=CUBE(side);`
- The preprocessor would expand the statement to  
`volume =(side*side*side)`

## Nesting of macros:

- We can also use one macro in the definition of another macro. That is macro definitions may be nested. Consider the following macro definitions

```
define SQUARE(x) ((x)*(x))
```

- **Undefining a macro:**
- A defined macro can be undefined using the statement  
`# undef identifier`
- This is useful when we want to restrict the definition only to a particular part of the program.



# File inclusion:

- The preprocessor directive "#include file name" can be used to include any file in to your program if the function s or macro definitions are present in an external file they can be included in your file
- In the directive the filename is the name of the file containing the required definitions or functions alternatively the this directive can take the form

```
#include< filename >
```

- Without double quotation marks. In this format the file will be searched in only standard directories.
- The c preprocessor also supports a more general form of test condition #if directive. This takes the following form

```
#if constant expression
{
 statement-1;
 statemet2'

}
#endif (Cont.)
```



- The constant expression can be a logical expression such as test < = 3 etc
- If the result of the constant expression is true then all the statements between the #if and #endif are included for processing otherwise they are skipped. The names TEST LEVEL etc., may be defined as macros.



# Module 2 Topics:

135

- **Programming OOPs**

- OOPs Introduction and Features
  - Class and Object
  - Functions
  - Constructor and Destructor
  - Types of Inheritance
  - Memory Management and Command line Arguments
  - Template
  - Working with files.

- **Data Structure**

- Algorithm and Flow Chart
- Addition, Subtraction, Multiplication and Division Algorithm
- Floating point Algorithm
- Decimal Arithmetic Unit
- Designing Algorithm Recursion
- Data Structure –Stack , Queue



# History

- C++ is an object-oriented extension of C
- C was designed by Dennis Ritchie at Bell Labs
- Used to write Unix, based on BCPL
- C++ designed by Bjarne Stroustrup at Bell Labs
- His original interest at Bell was research on simulation
- Early extensions to C are based primarily on Simula
- Called “C with classes” in early 1980’s
- Popularity increased in late 1980’s and early 1990’s
- Features were added incrementally
- Classes, templates, exceptions, multiple inheritance, type tests...



- Conventional programming, using high level language such as COBOL, Fortran and C commonly known as **procedure-oriented programming (POP)**.
- In the procedure oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculationg and printing.
- OOP treats data as a critical element in the program development and does not allow free Around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions.
- OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.
- Object oriented programming as an approach that way of modularizing programs by creating partitioned area for both data and functions that can be used as templates for creating copies of such modules on demand.

# Characteristics of OOP

- Emphasis is on data rather than procedure
- Programs are divided into what are known as objects
- Data and Functions are tied together in the data structure
- Data is hidden and can not be accessed by external functions
- Objects may communicate with each other through functions
- New data and functions can be easily added if necessary



# Features of Object – Oriented Programming

- Objects
- Classes
- Data abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message Passing



# Structure of a C++ Program

```
// my first program in C++

#include <iostream.h>
using namespace std;

int main ()
{
 cout << "Hello World!";
 return 0;
}
--- Output ---
Hello World!
```



**// my first program in C++**

- This is a comment line. All lines beginning with two slash signs (//) are considered comments

**#include <iostream>**

- In this case the directive #include <iostream> tells the preprocessor to include the iostream standard file.
- This specific file (iostream) includes the declarations of the basic standard input-output library in C++,

**using namespace std;**

- All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name std.

**int main ()**

- The main function is the point by where all C++ programs start their execution, independently of its location within the source code.



```
cout << "Hello World!";
```

- cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (which usually is the screen).

```
return 0;
```

- The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code 0).
- A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

```
int main ()
{
 cout << "Hello World!";
 return 0;
}
```

We could have written:

```
int main () { cout << "Hello World!"; return 0; }
```

- All in just one line and this would have had exactly the same meaning as the previous code.
- In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose.

# Comments

C++ supports two ways to insert comments:

```
// line comment single line
/* block comment */ multi line
```



# KEYWORDS

- Keywords are the reserved words in any language which have a special pre defined meaning and cannot be used for any other purpose. You cannot use keywords for naming variables or some other purpose.
- We saw the use of keywords main, include, return, int in our first C++ program.

# IDENTIFIERS

- Identifiers are the name of functions, variables, classes, arrays etc. which you create while writing your programs.
- Identifiers are just like names in any language. There are certain rules to name identifiers in C++.



# Identifiers Rules:

- Identifiers must contain combinations of digits, characters and underscore (\_).
- Identifier names cannot start with digit.
- Keyword cannot be used as an identifier name and upper case and lower case are distinct.
- Identifier names can contain any combination of characters as opposed to the restriction of 31 letters in C.

## CONSTANTS

- Constants are fixed values which cannot change. For example 123, 12.23, 'a' are constants.



# Declaration of variables

- Syntax:
  - Data\_type variable\_name
- For example:

```
int a;
float mynumber;
```

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

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/va>



## The scope of the Variable:

- ▶ Local Variables
- ▶ Global Variables

### Local Variables:

- The variables defined local to the block of the function would be accessible only within the block of the function and not outside the function. Such variables are called local variables.
- Scope of the local variables is limited to the function in which these variables are declared.

#### Refer this example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/localVar>



# Global Variables:

- Global variables are one which are visible in any part of the program code and can be used within all functions and outside all functions used in the program.
- The method of declaring global variables is to declare the variable outside the function or block.

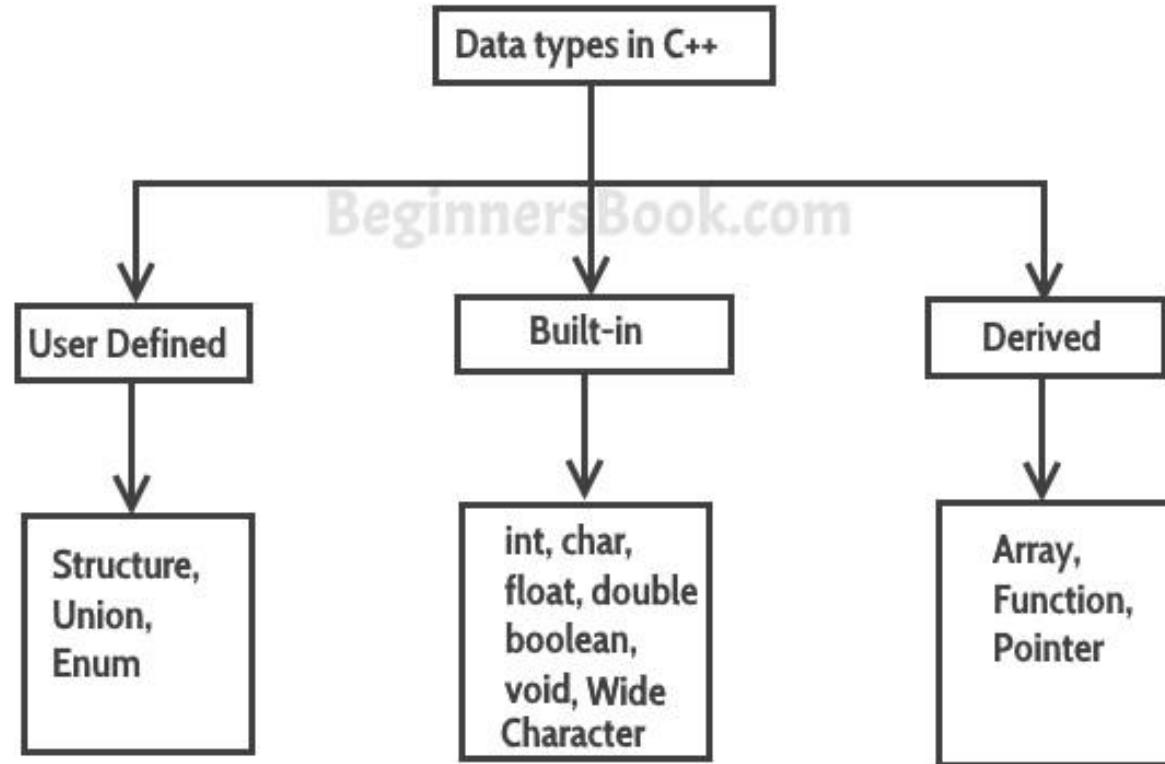
## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/globalVariable.cpp>



# Data Type

150



# Integer Data type

Integer Data Types

| Type               | Size(in bytes) | Range                     |
|--------------------|----------------|---------------------------|
| int                | 2              | -32768 to 32767           |
| signed int         | 2              | -32768 to 32767           |
| unsigned int       | 2              | 0 to 65535                |
| shortint           | 2              | -32768 to 32767           |
| signed short int   | 2              | -32768 to 32767           |
| unsigned short int | 2              | -32768 to 32767           |
| longint            | 4              | -2147483648 to 2147483647 |
| signed longint     | 4              | -2147483648 to 2147483647 |
| unsigned longint   | 4              | 0 to 4294967295           |

# Float Data type:

Floating Point Data Types

| Type        | Size(in bytes) | Range                                           | Digits of Precision |
|-------------|----------------|-------------------------------------------------|---------------------|
| float       | 4              | $3.4 \cdot 10^{-38}$ to $3.4 \cdot 10^{38}$     | 7                   |
| double      | 8              | $1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$   | 15                  |
| long double | 10             | $3.4 \cdot 10^{-4932}$ to $3.4 \cdot 10^{4932}$ | 18                  |

# Character data type

Character Data Types

| Type          | Size (in bytes) | Range        |
|---------------|-----------------|--------------|
| char          | 1               | - 128 to 127 |
| Signed char   | 1               | - 128 to 127 |
| unsigned char | 1               | 0 to 255     |



# Type Casting

The C-style casting takes the syntax as....

(type) expression

C++-style casting is as below namely:

type (expression)

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/typeCasting.cpp>



# C++ Storage Classes

**Automatic, External and Static.**

## What is Storage Class?

- Storage class defined for a variable determines the accessibility and longevity of the variable.
- The accessibility of the variable relates to the portion of the program that has access to the variable. The longevity of the variable refers to the length of time the variable exists within the program.

### Automatic:

- Variables defined within the function body are called automatic variables. **auto** is the keyword used to declare automatic variables.
- By default and without the use of a Keyword, the variables defined inside a function are automatic variables.



## External

- External variables are also called global variables.
- External variables are defined outside any function, memory is set aside once it has been declared and remains until the end of the program.
- These variables are accessible by any function. This is mainly utilized when a programmer wants to make use of a variable and access the variable among different function calls.

## Static

- The static automatic variables, as with local variables, are accessible only within the function in which it is defined.
- Static automatic variables exist until the program ends in the same manner as external variables. In order to maintain value between function calls, the static variable takes its presence.



# C++ Character Functions

- The C++ char functions are extremely useful for testing and transforming characters.
- These functions are widely used and accepted. In order to use character functions header file <cctype> is included into the program.
- Some of the commonly used character functions are:  
**isalnum()**- The function isalnum() returns nonzero value if its argument is either an alphabet or integer. If the character is not an integer or alphabet then it returns zero.  
**isalpha()**. The function isalpha() returns nonzero if the character is an uppercase or lower case letter otherwise it returns zero.  
**iscntrl()** - The function iscntrl() returns nonzero if the character is a control character otherwise it returns zero.  
**isdigit()**- The function isdigit() returns nonzero if the character is a digit, i.e. through 0 to 9. It returns zero for non digit character.

**isgraph()**- The function isgraph() returns nonzero if the character is any printable

character other than space otherwise it returns zero.

**islower()**- The function islower() returns nonzero for a lowercase letter otherwise it returns zero.

**isprint()**- The function isprint() returns nonzero for printable character including space otherwise it returns zero.

**isspace()**- The function isspace() returns nonzero for space, horizontal tab, newline

character, vertical tab, formfeed, carriage return; otherwise it returns zero.

**ispunct()**- The function ispunct() returns nonzero for punctuation characters otherwise it returns zero. The punctuation character excludes alphabets, digits and space.



**isupper()**- The function isupper() returns nonzero for an uppercase letter otherwise it returns zero.

**tolower()**- The function tolower() changes the upper case letter to its equivalent lower case letter. The character other than upper case letter remains unchanged.

**toupper()**- The function toupper() changes the lower case letter to its equivalent upper case letter. The character other than lower case letter remains unchanged.

**isxdigit()**- The function isxdigit() returns nonzero for hexadecimal digit i.e. digit

from 0 to 9, alphabet 'a' to 'f' or 'A' to 'F' otherwise it returns zero.



# C++ Functions



- A function is a block of code which only runs when it is called.
  - You can pass data, known as parameters, into a function.
  - Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.
- 
- **Refer This Example:**
  - <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Function/simpleFunction.cpp>

- A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

```
return_type function_name(parameter list)
{
 body of the function
}
```

- **Return Type:** The return\_type is the data type of the value the function returns. When function returns no value void is written
- **Function Name:** This is the actual name of the function.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:**
- The function body contains a collection of statements that define what the function does.

# Inline Functions

## What is Inline Function?

- Inline functions are functions where the call is made to inline functions. The actual code then gets placed in the calling program.

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Function/inline.cpp>



## C++ function call by value

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

### Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Function/callByValue.cpp>



## C++ function call by reference

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

### Refer this Example:

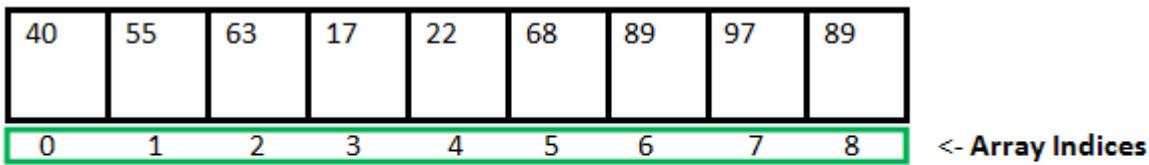
<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Function/callByReference.cpp>



# Array



- An array is a collection of several data items of the same data type stored in contiguous memory locations.



Array Length = 9  
First Index = 0  
Last Index = 8

- Array Declaration by specifying size:**

```
Datatype array_name[size]; int age [10];
```

- Array Declaration by initializing elements:**

```
int arr[] = { 10, 20, 30, 40 }
```

- Array declaration by specifying size and initializing elements**

```
int arr[5] = { 10, 20, 30, 40 }
```

# Refer This Example

<https://github.com/TopsCode/SoftwareEngineering/blob/master/c%2B%2B/Array/1dArray.cpp>



# MULTIDIMENSIONAL ARRAY

The multidimensional arrays are arrays of arrays. The general form of a multidimensional array is: -

Type `array_name[Size1][Size2]..[Size n];`

The two dimensional array can be declared as ...

`int age[2][5];`

This array has two rows and 5 columns. The three dimensional array can be declared as

`int age[2][3][5];`



- When an element of the array of n dimensional is referenced it uses n index values.
- For example first element of a two dimensional array declared above is referred as age[0][0] as the index starts from zero and last element is referred as age[1][4]. Here is a program which calculates the average of the elements of the row of the two dimensional array.

### Example of 2D array

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Array/2DArray.cpp>



# Null Terminated String

- The most common use of one dimensional array is for character strings.
- The null terminated string is a character array with a null character at the end. The characters form the string with a null character indicating the termination of the string. A null terminated string can be declared as...  
`char name[10];`
- It will hold 10 characters with a null at the end. The size of the array can be more than the length of the array. The array can be initialized as  
`char name[10]={'j','e','s','u','s'};`
- The first 5 elements are initialized and rest elements are null characters. Here is a program which calculates the length of the string.

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/String/length.cpp>

# OOPS

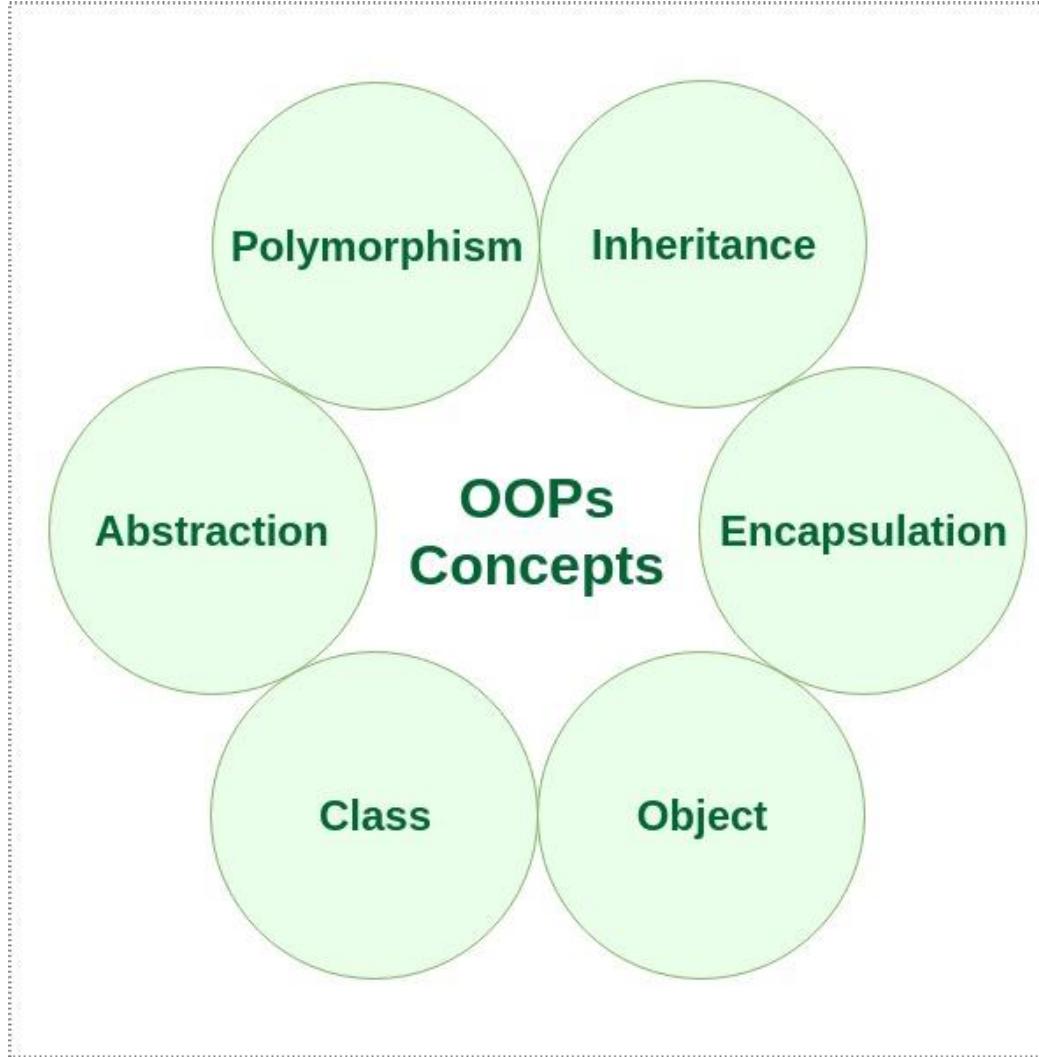


# Introduction

172

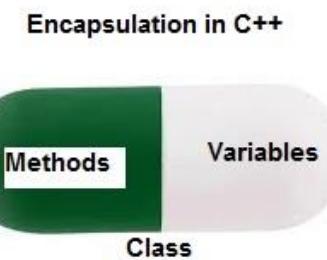
- Object-oriented programming is a paradigm in programming that represents real-life objects or entities in code,
- Some Language that supports OOPS:  
Java, J2EE, C++, C#, Visual Basic.NET, Python and JavaScript
- OOP is widely accepted as being far more flexible than other computer programming languages.
- The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.





## Encapsulation:

In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate



## Abstraction:

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.



## **Polymorphism:**

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

C++ supports operator overloading and function overloading.

## **Inheritance:**

- The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.
- Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.



## Classes

- These contain data and functions bundled together under a unit. In other words class is a collection of similar objects. When we define a class it just creates template or Skelton. So no memory is created when class is created. Memory is occupied only by object. for eg. :

Fruit is class of apple.

## Objects

In other words object is an instance of a class.

NOTE: In other words classes acts as data types for objects.

## Member functions

The functions defined inside the class as above are called member functions.

### Example:

```
class classname
{
 variable declarations;
 Data Functions;
};

main ()
{
 classname
 objectname1,objectname2,..;
}
```



# Types of accessifiers

| Private                        | Public                       | Protected                                    |
|--------------------------------|------------------------------|----------------------------------------------|
| Only for that class can access | Other class can also access. | Only immediate inheritance class can access. |

|                  | Direct-access scope        | Class/object scope |
|------------------|----------------------------|--------------------|
| Private          | Declaring class            | Declaring class    |
| Protected        | All derived classes        | Declaring class    |
| Friend           | Derived in-project classes | Declaring project  |
| Protected Friend | All derived classes        | Declaring project  |
| Public           | All derived classes        | All projects       |

Example of accessifiers:

```
Class classname
{
 private:
 datatype data;

 public:
 Member functions
};
main ()
{
 classname objectname1,objectname2,..;
}
```



# How to Access C++ Class Members

It is possible to access the class members after a class is defined and objects are created.

General syntax to access class member:

```
Object_name.function_name
(arguments);
```

```
class exforsys
{
 int a, b;
 public:
 void sum(int,int);
} e1;
e1.sum(5,6);
```



# Scope Resolution Operator

Member functions can be defined within the class definition or separately using **scope resolution operator**, `::`. Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier. So either you can define

```
class Box
{
public:
 double length; // Length of a box
 double breadth; // Breadth of a box
 double height; // Height of a box

 double getVolume(void)
 {
 return length * breadth * height;
 }
};
```



# Refer This Example

- ▶ **Simple Class:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/class1.cpp>
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/class2.cpp>



# Constructor

- It is a member function which initializes a class.
- A constructor has:
  - (i) the same name as the class itself
  - (ii) no return type

A constructor is called **automatically** invoked whenever a new instance of a class is created.

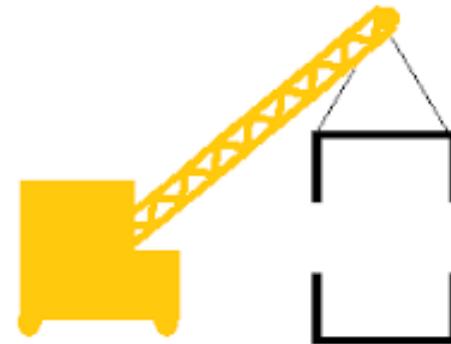
You must supply the arguments to the constructor when a new instance is created.

If you do not specify a constructor, the compiler generates a default constructor for you (expects no parameters and has an empty body).



## Types of Constructor

- ❑ Simple(Default) Constructor
- ❑ Parameterized Constructor
- ❑ Copy Constructor



**Constructor**

Bike b=new Bike();

# Refer This Example

► **Default Constructor:**

- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/DefaultConstructor.cpp>

► **Parameterised Constructor:**

- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/parameterisedConstructor.cpp>

► **Copy Constructor:**

- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructooor.cpp>



# Dynamic Constructors

- The constructor can also be used to allocate memory while creating objects. This will enable the system to allocate the right amount for each object when the objects are not of the same size, thus resulting in the saving of memory.
- Allocation of memory to objects at the time of their construction is known as dynamic constructor of objects. The memory is allocated with the help of the new operator.

## Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/DynamicConstructor.cpp>



# Virtual Constructor

- C++ being static typed (the purpose of RTTI is different) language, it is meaningless to the C++ compiler to create an object polymorphically.
- The compiler must be aware of the class type to create the object. In other words, what type of object to be created is a compile time decision from C++ compiler perspective. If we make constructor virtual, compiler flags an error. In fact except inline, no other keyword is allowed in the declaration of constructor.
- In practical scenarios we would need to create a derived class object in a class hierarchy based on some input. Putting in other words, object creation and object type are tightly coupled which forces modifications to extended. The objective of virtual constructor is to decouple object creation from it's type.

## Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Constructor/VirtualConstructor.cpp>

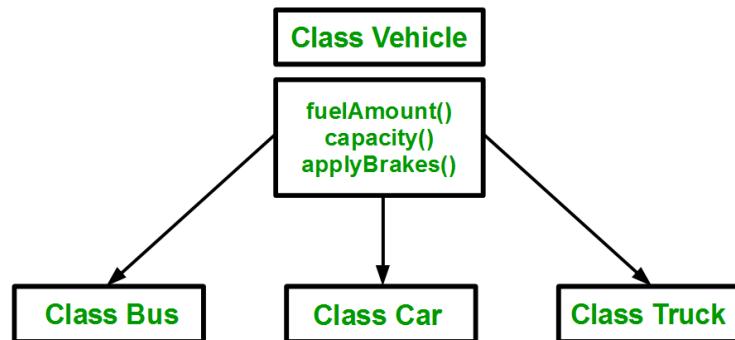


## What is a Destructor?

- It is a member function which deletes an object.
- A destructor function is called automatically when the object goes out of scope:
  - (1) the function ends
  - (2) the program ends
  - (3) a block containing temporary variables ends
  - (4) a delete operator is called
- A destructor has:
  - (i) the same name as the class but is preceded by a tilde (~)
  - (ii) no arguments and return no values

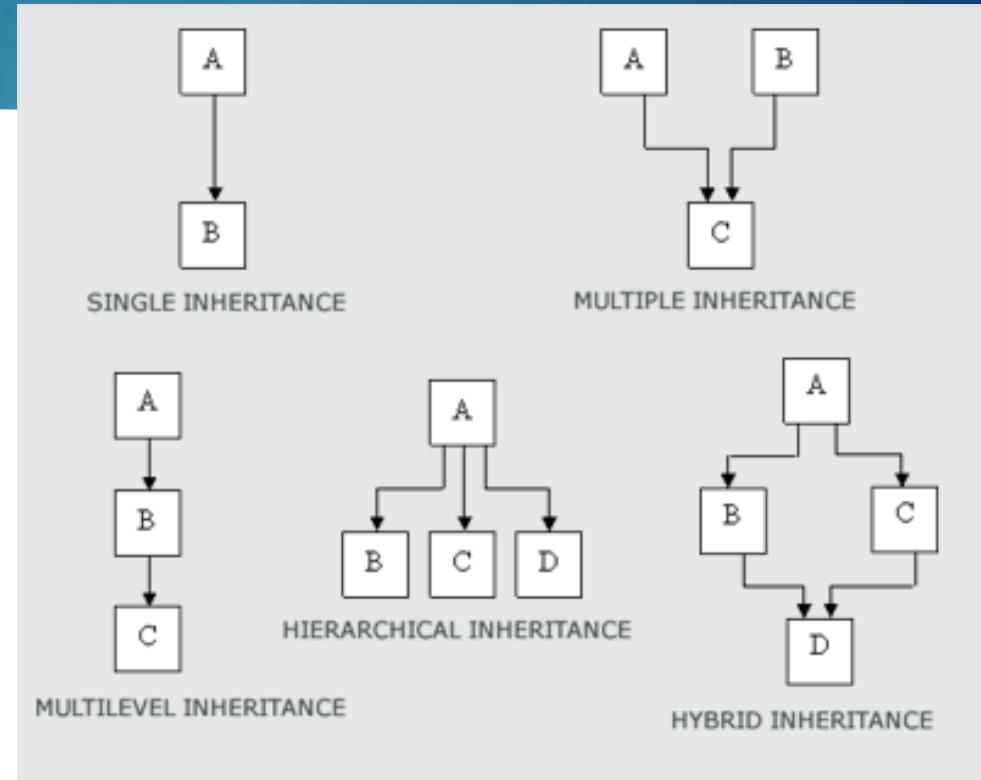
# Inheritance

- Inheritance is a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.



# Types of Inheritance

- Single Inheritance
- Multiple Inheritcance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance



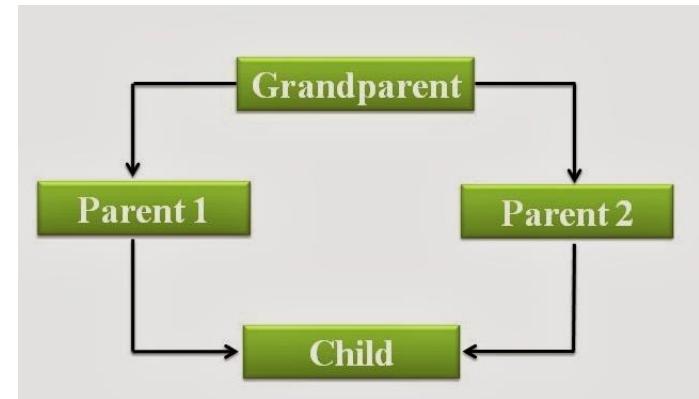
# Refer this Example

- ▶ **Single Inheritance:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Inheritance/single.cpp>
- ▶ **Multiple Inheritance:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Inheritance/multiple.cpp>
- ▶ **Multilevel Inheritance:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Inheritance/multilevel.cpp>
- ▶ **Hierarchical Inheritance**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Inheritance/hierarchical>
- ▶ **Hybrid Inheritance**



# Virtual Base Classes

- Virtual base class is used in situation where a derived have multiple copies of base class.
- When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class.



**Refer This Example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/virtualC...>

# Constructor in Derived Class

- Base class constructors are automatically called for you if they have no argument.
- If you want to call a superclass constructor with an argument, you must use the subclass's constructor initialization list.
- Unlike Java, **C++ supports multiple inheritance** (for better or worse), so the base class must be referred to by name, rather than "super()".
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/ConstructorInheritance.cpp>

# The this pointer (C++ only)

- Every object in C++ has access to its own address through an important pointer called this pointer.
- The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/thisPointer.cpp>

# Polymorphism and Overloading



- Poly refers many.
- "single interface having multiple implementations."
- That is making a function or operator to act in different forms depending on the place they are present is called Polymorphism. Overloading is a kind of polymorphism.
- 2 Types of polymorphism:
  - **Static** : compile time
  - **Dynamic** : run time



# Static Polymorphism

- ▶ 2 Types:
- ▶ **Function overloading** which is the process of using the same name for two or more functions.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/FunctionOverloading/example1.cpp>

- ▶ **Operator overloading** which is the process of using the same operator for two or more operands.



# Refer This Example

- ▶ **Unary Operator Overloading:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/OperatorOverloading/Unary.cpp>
  
- ▶ **Binary Operator Overloading:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/OperatorOverloading/binary.c>

# Dynamic Polymorphism

- This refers to the entity which changes its form depending on circumstances at runtime.

## Virtual Function:

- A virtual function can be defined as the member function within a base class which you expect to redefine in derived classes.
- For creating a virtual function, you have to precede your function's declaration within the base class with a **virtual** keyword.
- **Refer This Example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Function/virtualFunction.c>



# What is Pure Virtual Function

Pure Virtual Function is a Virtual function with no body.

```
class classname //This denotes the base class of C++ virtual
function
{
 public:
 virtual void virtualfunctionname() = 0 //This denotes the
pure
 virtual function in
C++
};
```

## Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Function/pureVirtualFunction.cpp>



# Friend Function

- ▶ A friend function is used for accessing the non-public members of a class.
- ▶ A class can allow non-member functions and other classes to access its own private data, by making them friends. Thus, a friend function is an ordinary function or a member of another class.

## How to define and use Friend Function in C++:

- The friend function is written as any other normal function, except the function declaration of these functions is preceded with the keyword friend.
- The friend function must have the class to which it is declared as friend passed to it in argument.



## Some important points to note while using friend functions in C++:

- The keyword friend is placed only in the function declaration of the friend function and not in the function definition.
- ▶ It is possible to declare a function as friend in any number of classes.
- ▶ When a class is declared as a friend, the friend class has access to the private data of the class that made this a friend.
- ▶ A friend function, even though it is not a member function, would have the rights to access the private members of the class.
- ▶ It is possible to declare the friend function as either private or public.
- ▶ The function can be invoked without the use of an object.
- ▶ **Refer This Example:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Function/FriendFunction.cpp>

- An abstract class is a class that is designed to be specifically used as a base class.
- An abstract class contains at least one pure virtual function.
- You declare a pure virtual function by using a pure specifier (= 0) in the declaration of a virtual member function in the class declaration.
- You cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class.
- Refer This Example:
  - <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Abstract%2C.cpp>

# Static in C++ Class

- We can define class members static using **static** keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only **one copy of the static member.**
- A static member is **shared by all objects of the class**. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c++/OOPS/static/static.c>

# Static Function Members:

204

- By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.
- A static member function can only access static data member, other static member functions and any other functions from outside the class.
- Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.
- **Refer this Example:**  
<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/static/staticFunction.c>



# File in C++

- C++ provides the following classes to perform output and input of characters to/from files:
- **ofstream**: Stream class to write on files
- **ifstream**: Stream class to read from files
- **fstream**: Stream class to both read and write from/to files.

## Operations in File Handling:

- Creating a file: open()
- Reading data: read()
- Writing new data: write()
- Closing a file: close()



## Opening a File:

- A file must be opened before you can read from it or write to it. Either the ofstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only.
- Syntax:

```
void open(const char *filename, ios::openmode mode);
```

| Sr.No | Mode Flag & Description                                                                                  |
|-------|----------------------------------------------------------------------------------------------------------|
| 1     | <b>ios::app</b><br>Append mode. All output to that file to be appended to the end.                       |
| 2     | <b>ios::ate</b><br>Open a file for output and move the read/write control to the end of the file.        |
| 3     | <b>ios::in</b><br>Open a file for reading.                                                               |
| 4     | <b>ios::out</b><br>Open a file for writing.                                                              |
| 5     | <b>ios::trunc</b><br>If the file already exists, its contents will be truncated before opening the file. |

## Writing to a File:

- For writing in a file we use **ios::out**.
- Syntax
  - `FilePointer << " Lines"`

## Reading from a File:

- To read from a file use **ios::in**.
- information from the output file is obtained with the help of following syntax
  - `FilePointer >> variable`

## Refer This Example:

[https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/File/Read\\_Write.cpp](https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/File/Read_Write.cpp)



# File Position Pointers:

- Both **istream** and **ostream** provide member functions for repositioning the file-position pointer.
- These member functions are **seekg** ("seek get") for istream and **seekp** ("seek put") for ostream.
- `seekg()` - for get pointer
- `seekp()` - for put pointer
- These functions take two arguments:
  - The first argument is the relative offset i.e. the number of bytes the file pointer has to be moved. (+ for forward and - for backward.).
  - The second argument is the position of the file pointer from where the offset is to be considered. The default argument for this is the `beg` (beginning of the file). It can take `ios::beg` (beginning), `ios::end` (end of file), and `ios::cur` (current pointer position).

# Refer This Example

- ▶ Seek:
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/File/seek.cpp>



**C++ provides following two types of string representations:**

1. The C-style character string.
2. The string class type introduced with Standard C++.

**Refer This Example:**

C-style String:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/String/cString.cpp>

## C++ supports a wide range of functions that manipulate null-terminated strings:

211

| S.N. | Function & Purpose                                                                                            |
|------|---------------------------------------------------------------------------------------------------------------|
| 1    | <b>strcpy(s1, s2);</b><br>Copies string s2 into string s1.                                                    |
| 2    | <b>strcat(s1, s2);</b><br>Concatenates string s2 onto the end of string s1.                                   |
| 3    | <b>strlen(s1);</b><br>Returns the length of string s1.                                                        |
| 4    | <b>strcmp(s1, s2);</b><br>Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5    | <b>strchr(s1, ch);</b><br>Returns a pointer to the first occurrence of character ch in string s1.             |
| 6    | <b>strstr(s1, s2);</b><br>Returns a pointer to the first occurrence of string s2 in string s1.                |

# Refer This Example

- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/String/cStringFunction.cpp>



# The String Class in C++:

- The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality. We will study this class in C++ Standard Library but for now let us check following example:
- At this point you may not understand this example because so far we have not discussed Classes and Objects. So can have a look and proceed until you have understanding on Object Oriented Concepts.
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/String/stringClass.cpp>

## Need for Memory Management operators

- The concept of arrays has a block of memory reserved. The disadvantage with the concept of arrays is that the programmer must know, while programming, the size of memory to be allocated in addition to the array size remaining constant.

## What are memory management operators?

- There are two types of memory management operators in C++:
  - new
  - delete



## New operator:

- The new operator in C++ is used for dynamic storage allocation. This operator can be used to create object of any type.

**Syntax:** pointer variable = new datatype;

**Example**

```
int *a=new
int;
```

## Delete Operator:

The delete operator in C++ is used **for releasing memory space** when the object is no longer needed. Once a new operator is used, it is efficient to use the corresponding delete operator for release of memory.

**Syntax:** pointer variable;  
delete  
pointer variable;  
delete  
a;



# Refer This Example

- ▶ **New And Delete Operator:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Memory%20Management%20Operator/Example1.cpp>



# Templates

- Templates are powerful features of C++ which allows you to write generic programs.
- In simple terms, you can create a single function or a class to work with different data types using templates.
- Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

The concept of templates can be used in two different ways:

1. Function Templates
2. Class Templates



# Function Template

- ▶ Function Template can work with different data types at once.
- ▶ A function template defines a family of functions.
- ▶ **Syntax:**
  - ▶ `template <class type> ret-type func-name(parameter list) {`
  - ▶ `// body of function`
  - ▶ `}`
- ▶ **Refer This Example:**
  - ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Template/functionTemplate.cpp>

# Class Template

- ▶ Class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.
- ▶ A class template defines a family of classes.
  
- ▶ Syntax:
- ▶ `template <class type> class class-name {`
- ▶ `.`
- ▶ `.`
- ▶ `.`
- ▶ `}`

# Refer This Example

- ▶ **Template Class:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Template/templateClass.cpp>
  
- ▶ **Function Template with Multiple Parameter:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Template/templateFunction.cpp>



# Command Line Argument

- In C & C++, it is possible to accept command line arguments.
- To use command line argument , main function accepts two arguments:
  - one argument is number of command line arguments,
  - and the other argument is a full list of all of the command line arguments.
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Command%20Line%20Argument/cmd.cpp>



# Data Structure And Algorithms



- An algorithm is a sequence of steps to solve a particular problem or algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.
- Algorithm has the following characteristics
  - **Input:** An algorithm may or may not require input
  - **Output:** Each algorithm is expected to produce at least one result
  - **Definiteness:** Each instruction must be clear and unambiguous.
  - **Finiteness:** If the instructions of an algorithm are executed, the algorithm should terminate after finite number of steps

# HOW TO WRITE ALGORITHMS:

224

**Step 1: Define your algorithms input:** Many algorithms take in data to be processed, e.g. to calculate the area of rectangle input may be the rectangle height and rectangle width.

**Step 2 Define the variables:** Algorithm's variables allow you to use it for more than one place. We can define two variables for rectangle height and rectangle width as HEIGHT and WIDTH (or H & W). We should use meaningful variable name e.g. instead of using H & W use HEIGHT and WIDTH as variable name.

**Step 3 Outline the algorithm's operations:** Use input variable for computation purpose, e.g. to find area of rectangle multiply the HEIGHT and WIDTH variable and store the value in new variable (say) AREA. An algorithm's operations can take the form of multiple steps and even branch, depending on the value of the input variables.

**Step 4 Output the results of your algorithm's operations:** In cc rectangle output will be the value stored in variable AREA. if variables described a rectangle with a HEIGHT of 2 and a WIDTH of 3, the algorithm would output the value of 6.



# FLOWCHART

- ▶ A flowchart is a graphical representation of an algorithm.
- ▶ Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing

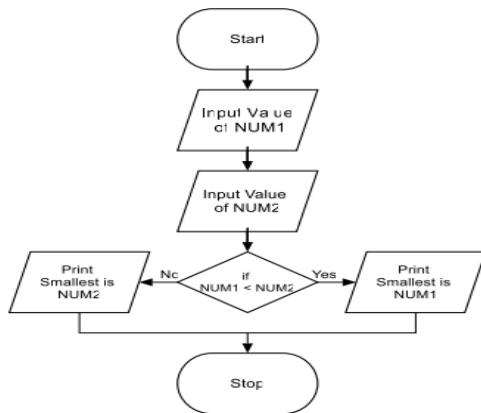
The process of drawing a flowchart for an algorithm is known as “**flowcharting**”.

#### **Algorithm & Flowchart to find the smallest of two numbers**

```

Algorithm
Step-1 Start
Step-2 Input two numbers say
NUM1,NUM2
Step-3 IF NUM1 < NUM2 THEN
 print smallest is NUM1
ELSE
 print smallest is NUM2
ENDIF
Step-4 Stop

```



# Algorithm And Flowchart Example

## Algorithm & Flowchart to find the smallest of two numbers

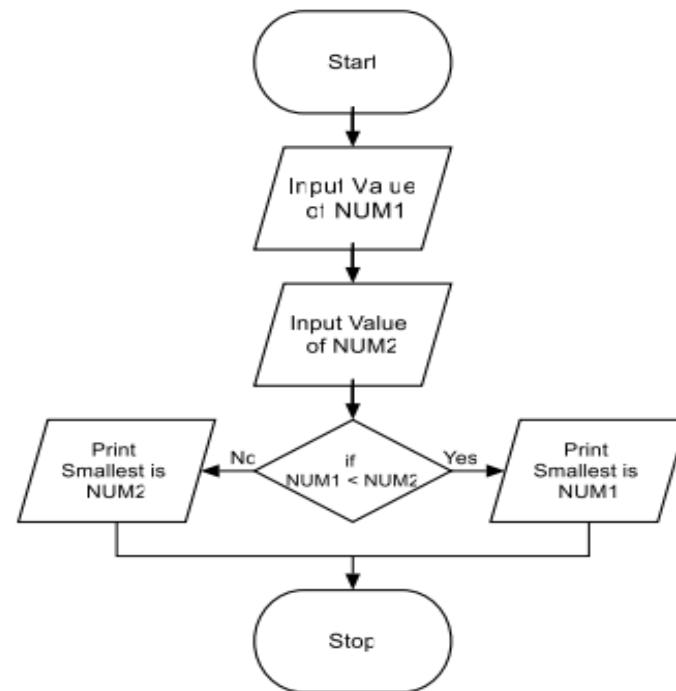
### Algorithm

Step-1 Start

Step-2 Input two numbers say  
NUM1,NUM2

Step-3 IF NUM1 < NUM2 THEN  
    print smallest is NUM1  
ELSE  
    print smallest is NUM2  
ENDIF

Step-4 Stop



# Binary Number

- ▶ Binary Numbers are the flow of information in the form of zeros and ones used by digital computers and systems.
- ▶ The binary number system is an alternative to the decimal (10-base) number system that we use every day.
- ▶ Binary numbers are important because using them instead of the decimal system simplifies the design of computers and related technologies.
- ▶ The simplest definition of the binary number system is a system of numbering that uses only two digits—0 and 1—to represent numbers, instead of using the digits 1 through 9 plus 0 to represent numbers.

# Conversion from Decimal to Binary

|   |    |
|---|----|
| 2 | 25 |
| 2 | 12 |
| 2 | 6  |
| 2 | 3  |
| 2 | 1  |
|   | 0  |

1 ← First remainder  
 0 ← Second Remainder  
 0 ← Third Remainder  
 1 ← Fourth Remainder  
 1 ← Fifth Reaminder

Read Up

Binary Number = 11001

Circuit Globe

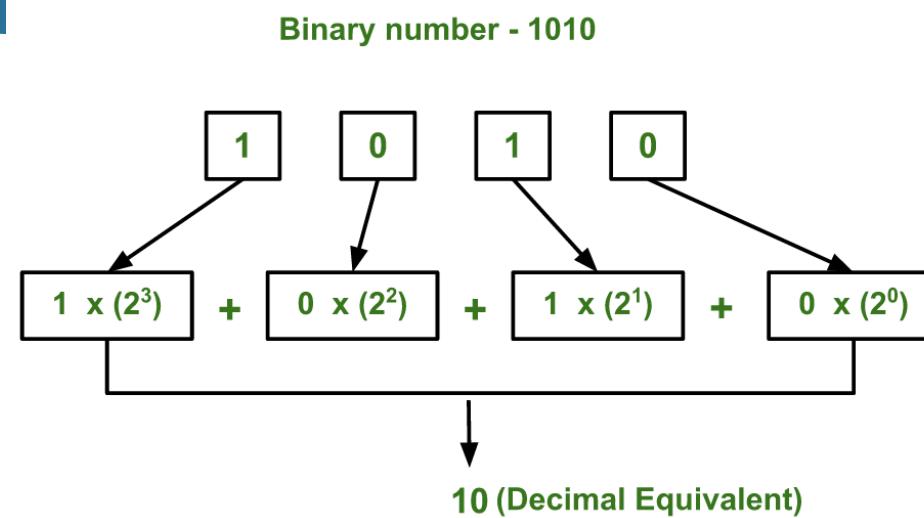
Decimal number : 17

|   |    |   |
|---|----|---|
| 2 | 17 | 1 |
| 2 | 8  | 0 |
| 2 | 4  | 0 |
| 2 | 2  | 0 |
|   | 1  |   |

Binary number: 10001

1. Store the remainder when the number is divided by 2 in an array.
2. Divide the number by 2
3. Repeat the above two steps until the number is grec zero.
4. Print the array in reverse order now.

# Conversion from Binary to Decimal



# Computer Arithmetic

► It involves:

- Addition
- Subtraction
- Multiplication and
- Division



Two ways by which negative numbers are represented:

- ▶ 1. Sign Magnitude
- ▶ 2. 2's complement method



# Sign magnitude

- ▶ It is a very simple representation of negative numbers.
- ▶ In sign magnitude the first bit is dedicated to represent the sign and hence it is called sign bit.
- ▶ Sign **bit '1'** represents **negative** sign.
- ▶ Sign **bit '0'** represents **positive** sign.

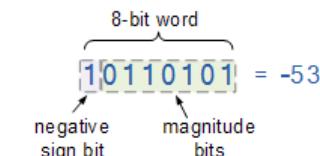
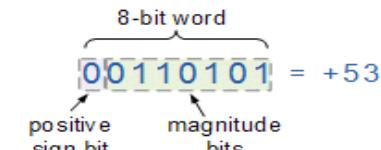
For example,

- ▶  $+25 = 011001$  [Where 11001 = 25 And 0 for '+']
- ▶  $-25 = 111001$  [Where 11001 = 25 And 1 for '-'.]

- ▶ But there is one problem in sign magnitude and that is two representations of 0

$$+0 = 000000$$

$$-0 = 100000$$



## 2's complement method

- ▶ To represent a negative number in this form, first we need to take the 1's complement of the number represented in simple positive binary form and then add 1 to it.
- ▶ For example:

$$710 = 01112$$

1's complement of 0111 = 1000

Adding 1 to it,  $1000 + 1 = 1001$

So,  $-710 = 10012$



# Addition

- ▶ The binary addition of two bits (a and b) is defined by the table:

| <u>a</u> | <u>b</u> | <u><math>a + b</math></u> |
|----------|----------|---------------------------|
| 0        | 0        | 0                         |
| 0        | 1        | 1                         |
| 1        | 0        | 1                         |
| 1        | 1        | 0                         |

carry 0                            carry 1

- When adding n-bit values, the values are added in corresponding bit-wise pairs, with each carry being added to the next most significant pair of bits. The same algorithm can be used when adding pairs of unsigned or pairs of signed values.

# 4-bit addition example

$$\begin{array}{r}
 \text{value 1:} & 0 & 1 & 0 & \leftarrow \text{carry values} \\
 + \text{value 2:} & 1 & 0 & 1 & 1 \\
 \hline
 \text{result:} & + & 0 & 0 & 1 & 0
 \end{array}$$

} bit-wise pairs

4-Bit Example B:

$$\begin{array}{r}
 \text{value 1:} & 1 & 1 & 1 & 0 & \leftarrow \text{carry values} \\
 + \text{value 2:} & 1 & 0 & 1 & 1 \\
 \hline
 \text{result:} & + & 0 & 1 & 1 & 0 \\
 & & & | & & \\
 & & & 1 & 0 & 0 & 1
 \end{array}$$

} bitwise pairs

↑ ignored      ↓ 4-bit result

Since computers are constrained to deal with fixed-width binary values, any carry out of the most significant bit-wise pair is ignored.

# Validity of result:

- ▶ First consider the case where the binary values are intended to represent unsigned integers (i.e. counting numbers).
- ▶ Adding the binary values representing two unsigned integers will give the correct result (i.e. will yield the binary value representing the sum of the unsigned integer values) providing the operation does not overflow – i.e. when the addition operation is applied to the original unsigned integer values.
- ▶ The result is an unsigned integer value that is inside of the set of unsigned integer values that can be represented using the specified number of bits (i.e. the result can be represented using the fixed-width constraints imposed by the representation).

# Overflow

- When the values added in above example are considered as unsigned values, then the 4-bit result does not accurately represent the sum of the unsigned values.

$$\begin{array}{r} \text{value 1: } 11_{10} \\ + \text{value 2: } 6_{10} \\ \hline \text{result: } 17_{10} \end{array} \quad \begin{array}{r} 1110 \leftarrow \text{carry values} \\ 1011 \\ + 0110 \\ \hline 0001 \end{array} \quad \text{???? } 11 + 6 = 1 \text{ ?????}$$

- In this case, the operation has resulted in overflow: the result (17) is outside the set of values that can be represented using 4-bit binary number system values (i.e. 17 is not in the set {0, ..., 15}).
- The result (00012) is correct according to the rules for performing binary addition using fixed-width values, but truncating the carry out of the most significant bit resulted in the loss of information that was important to the encoding being used. If the carry had been kept, then the 5-bit result (100012) would have represented the unsigned integer sum correctly.

# Overflow

- ▶ In arithmetic an overflow happens when
  - ▶ The **sum of two positive numbers exceeds** the maximum positive value that can be represented using n bits:  $2^n - 1 - 1$
  - ▶ The **sum of two negative numbers falls below** the minimum negative value that can be represented using n bits:  $-2^n - 1$

# Example

- ▶ Four-bit arithmetic:
  - ▶ Sixteen possible values
  - ▶ Positive overflow happens when result  $> 7$
  - ▶ Negative overflow happens when result  $< -8$
- ▶ Eight-bit arithmetic:
  - ▶ 256 possible values
  - ▶ Positive overflow happens when result  $> 127$
  - ▶ Negative overflow happens when result  $< -128$

## Eight Conditions for Signed-Magnitude Addition/Subtraction

| Operation          | ADD Magnitudes | SUBTRACT Magnitudes |             |             |
|--------------------|----------------|---------------------|-------------|-------------|
|                    |                | $A > B$             | $A < B$     | $A = B$     |
| 1<br>$(+A) + (+B)$ | $+ (A + B)$    |                     |             |             |
| 2<br>$(+A) + (-B)$ |                | $+ (A - B)$         | $- (B - A)$ | $+ (A - B)$ |
| 3<br>$(-A) + (+B)$ |                | $- (A - B)$         | $+ (B - A)$ | $+ (A - B)$ |
| 4<br>$(-A) + (-B)$ | $- (A + B)$    |                     |             |             |
| 5<br>$(+A) - (+B)$ |                | $+ (A - B)$         | $- (B - A)$ | $+ (A - B)$ |
| 6<br>$(+A) - (-B)$ | $+ (A + B)$    |                     |             |             |
| 7<br>$(-A) - (+B)$ | $- (A + B)$    |                     |             |             |
| 8<br>$(-A) - (-B)$ |                | $- (A - B)$         | $+ (B - A)$ | $+ (A - B)$ |

# Subtraction

- ▶ The binary subtraction of two bits (a and b) is defined by the table:

| $a$ | $b$ | $a - b$ |
|-----|-----|---------|
| 0   | 0   | 0       |
| 1   | 0   | 1       |
| 1   | 1   | 0       |
| 0   | 1   | 1       |

borrow 0  
borrow 1

- ▶ When subtracting n-bit values, the values are subtracted in corresponding bit-wise pairs, with each borrow rippling down from the more significant bits as needed.
- ▶ If none of the more significant bits contains a 1 to be borrowed, then 1 may be borrowed into the most significant bit.

# Example

$$\begin{array}{r}
 1010 \\
 -0001 \\
 \hline
 \end{array}$$

must borrow from second digit

|          |          |                 |  |
|----------|----------|-----------------|--|
| becomes: | 0        | Interpretations |  |
|          | unsigned | signed          |  |
|          | 10       | -6              |  |
|          | -1       | -7              |  |
|          | 9        |                 |  |

no overflow in either case

$$\begin{array}{r}
 0 \\
 -1 \\
 \hline
 \end{array}$$

|          |          |                 |  |
|----------|----------|-----------------|--|
| becomes: | 1        | Interpretations |  |
|          | unsigned | signed          |  |
|          | 1        | 1               |  |
|          | -15      | -1              |  |
|          | 2        | 2               |  |

**Example**

borrow from above most signif. bit

$$\begin{array}{r}
 10101010101 \\
 -111111 \\
 \hline
 0010
 \end{array}$$

overflow in unsigned case  
no overflow in signed case



# When Overflow Occurs?

- ▶ For **unsigned values**, a carry out of (or a borrow into) the most significant bit indicates that overflow has occurred.
- ▶ For **signed values**, overflow has occurred when the sign of the result is impossible for the signs of the values being combined by the operation. For example, overflow has occurred if:
  - ▶ Two positive values are added and the sign of the result is negative
  - ▶ a negative value is subtracted from a positive value and the result is negative (a positive minus a negative is the same as a positive plus a positive, and should result in a positive value, i.e.  $a - (-b) = a + b$  )

**Adding a positive and a negative value will never overflow.** Reason: picture the two values on a number line as shown below. Suppose that a is a negative value, and b is a positive value. Adding the two values  $a + b$  will result in c such that c will always lie between a and b on the number line. If a and b can be represented precisely, then c can also be represented, and overflow will never occur.

# Multiplication

- ▶ Multiplication is a slightly more complex operation than addition or subtraction. Multiplying two n-bit values together can result in a value of up to  $2n$ -bits.
- ▶ There is a reasonably simple algorithm for multiplying binary values that represent unsigned integers, but the same algorithm cannot be applied directly to values that represent signed values (this is different from addition and subtraction where the same algorithms can be applied to values that represent unsigned or signed values!).
- ▶ Overflow is not an issue in n-bit unsigned multiplication, proving that  $2n$ -bits of results are kept.

# Decimal multiplication

$$\begin{array}{r} \text{(carry) } 1 \\ 37 \\ \times 12 \\ \hline 74 \\ 370 \\ \hline 444 \end{array}$$

## ► What are the rules?

- Successively multiply the multiplicand by each digit of the multiplier starting at the right shifting the result left by an extra left position each time each time but the first
- Sum all partial results



# Binary multiplication

$$\begin{array}{r} \text{(carry)} \\ 111 \\ 110\bar{1} \\ \times 101 \\ \hline 1101 \\ 00 \\ \hline 110100 \\ 1000001 \end{array}$$

## ► What are the rules?

► Successively multiply the multiplicand by each digit of the multiplier starting at the right shifting the result left by an extra left position each time each time but the first

► Sum all partial results



# Binary multiplication table

|          |   |   |
|----------|---|---|
| <b>X</b> | 0 | 1 |
| 0        | 0 | 0 |
| 1        | 0 | 1 |



# Multiplication Algorithm for Unsigned Number:

The following **shift-and-add algorithm** can be used to calculate the product of unsigned values:

```
unsigned int a;
```

```
unsigned int b;
```

```
unsigned long int sum; // need twice as many bits for result!
```

```
unsigned longint ashifted;
```

```
// calculate a * b
```

```
sum = 0;
```

```
ashifted = a;
```

```
for(i = 0 ; i < n ; i + +)
```

```
{ if (bi == 1) { sum+= ashifted; }
```

```
 ashifted = shift_left(ashifted) // shift_left function shifts value one
bit
```

```
}
```

```
// at this point, sum holds the product!
```



# Explanations

- ▶ The variable `ashifted` represents the value of the term  $a * 2^i$ .
- ▶ Each time through the loop, if the value of bit  $i$  is 1 then the value of the  $i$ th term is added (accumulated) to the variable `sum`, and the value of the next ( $i + 1$ th) term is calculated by shifting `ashifted` left by one bit (assume that during the shift operation a 0 is injected as the least significant bit).



# Binary multiplication

$$\begin{array}{r} 1101 \\ \times 101 \\ \hline 1101 \\ 00 \\ \hline 110100 \\ \hline 1000\textcolor{red}{1}\textcolor{green}{0}1 \end{array}$$

- ▶ Observe that the least significant bit added during each cycle remains unchanged

# Booth Multiplication Algorithm

- ▶ Booth's Algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.
- ▶ Example  $(2) * (-4) = 0010 * 1100$
- ▶ It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight  $2^k$  to weight  $2^m$  can be treated as  $2^{(k+1)}$  to  $2^m$ .
- ▶ Step1: Making the booth table:
  1. From the two numbers, pick the number with the smallest difference between a series of consecutive numbers, and make it a multiplier.

i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change ,so there are two changes on this one

1100 -- From 1 to 1 no change, 1 to 0 one change, 0 to 0 no change, so there is only one change on this one.

Therefore, multiplication of  $2 \times (-4)$ , where  $2_{\text{ten}}$  ( $0010_{\text{two}}$ ) is the r and  $(-4)_{\text{ten}}$  ( $1100_{\text{two}}$ ) is the multiplier.

2.

Let  $X = 1100$  (multiplier)

Let  $Y = 0010$  (multiplicand)

Take the 2's complement of Y and call it  $-Y$

$$-Y = 1110$$

3. Load the X value in the table.

4. Load 0 for  $X-1$  value it should be the previous first least significant bit of X

5. Load 0 in U and V rows which will have the product of X and Y at the end of operation.

6. Make four rows for each cycle; this is because we are multiplying four bits numbers.

| U    | V    | X    | X-1 |
|------|------|------|-----|
| 0000 | 0000 | 1100 | 0   |
|      |      |      |     |
|      |      |      |     |
|      |      |      |     |

Load the value  
1<sup>st</sup> cycle  
2<sup>nd</sup> cycle  
3<sup>rd</sup> Cycle  
4<sup>th</sup> Cycle

## Step 2: Booth Algorithm

Look at the first least significant bits of the multiplier “X”, and the previous least significant bits of the multiplier “X - 1”.

a. 0 0 Shift only 1 1

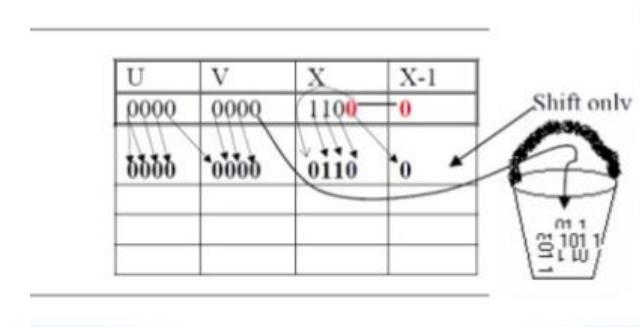
Shift only. 0 1

Add Y to U, and shift

1 0 Subtract Y from U, and shift or add (-Y)  
to U and shift

b. Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.

c. Shift X circular right shift because this will prevent us from using two registers for the X value.



Repeat the same steps until the four cycles are completed.

| U    | V    | X    | X-1 |
|------|------|------|-----|
| 0000 | 0000 | 1100 | 0   |
| 0000 | 0000 | 0110 | 0   |
| 0000 | 0000 | 0011 | 0   |
|      |      |      |     |
|      |      |      |     |

← Shift only

| U    | V    | X    | X-1 |
|------|------|------|-----|
| 0000 | 0000 | 1100 | 0   |
| 0000 | 0000 | 0110 | 0   |
| 0000 | 0000 | 0011 | 0   |
| 1110 | 0000 | 0011 | 0   |
| 1111 | 0000 | 1001 | 1   |
|      |      |      |     |

← Add -Y (0000 + 1110 = 1110)  
← Shift

| U           | V           | X           | X-1      |
|-------------|-------------|-------------|----------|
| 0000        | 0000        | 1100        | 0        |
| 0000        | 0000        | 0110        | 0        |
| 0000        | 0000        | 0011        | 0        |
| 1110        | 0000        | 0011        | 0        |
| 1111        | 0000        | 1001        | 1        |
| <b>1111</b> | <b>1000</b> | <b>1100</b> | <b>1</b> |

We have finished four cycles, so the answer is shown, in the last rows of U and V which is:  
11111000

# Division

- ▶ A division algorithm provides a **quotient** and a **remainder** when we divide two number.
- ▶ They are generally of two type slow algorithm and fast algorithm. Slow division algorithm are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.
- ▶ Result must verify the equality

$$\text{Dividend} = \text{Multiplier} \times \text{Quotient} + \text{Remainder}$$

TOPS TECHNOLOGIES PVT. LTD.  
SEP\_2019

# Decimal division (long division

$$\begin{array}{r} 303 \\ 7 \overline{)2126} \\ -210 \\ \hline 26 \\ -21 \\ \hline 5 \end{array}$$

- ▶ What are the rules?
- ▶ Repeatedly try to subtract smaller multiple of divisor from dividend
- ▶ Record multiple (or zero)
- ▶ At each step, repeat with a lower power of ten
- ▶ Stop when remainder is smaller than divisor

# Binary division

$$\begin{array}{r} 011 \\ 11 \overline{)1011} \\ -11 \\ \hline 1011 \\ > \underline{-11} \\ 101 \\ >> \underline{-11} \\ 10 \end{array}$$

x

► What are the rules?

- Repeatedly try to subtract powers of two of divisor from dividend
- Mark 1 for success, 0 for failure
- At each step, shift divisor one position to the right
- Stop when remainder is smaller than divisor

# Same division in decimal

$$\begin{array}{r} 2+1=3 \\ \hline 3 \overline{)11} \\ -\cancel{1}2 \\ \hline 11 \\ >\cancel{-}6 \\ \hline 5 \\ >\cancel{-}3 \\ \hline 2 \end{array}$$

x

► What are the rules?

- Repeatedly try to subtract powers of two of divisor from dividend
- Mark 1 for success, 0 for failure
- At each step, shift divisor one position to the right
- Stop when remainder is smaller than divisor

# Observations

- ▶ Binary division is actually simpler
  - ▶ We start with a left-shifted version of divisor
  - ▶ We try to subtract it from dividend
    - ▶ No need to find out which multiple to subtract
  - ▶ We mark 1 for success, 0 for failure
  - ▶ We shift divisor one position left after every attempt

# Issues with Binary Division

The implementation of division in a computer raises several practical issues:

- ▶ For integer division there are two results: the quotient and the remainder.
- ▶ The operand sizes (number of bits) to be used in the algorithm must be considered (i.e. the sizes of the dividend, divisor, quotient and remainder).
- ▶ Overflow was not an issue in unsigned multiplication, but is a concern with division.
- ▶ As with multiplication, there are differences in the algorithms for signed vs. unsigned division.



# Shift-and-subtract algorithm

The algorithm is the inverse of the shift-and-add multiplication algorithm

```
unsigned int d; // divisor
unsigned int quotient;
unsigned longint remainder;
unsigned longint dividend; // dividend has twice the number of bits
unsigned longint dshifted;

// calculate dividend / divisor = quotient & remainder
quotient = 0;
remainder = dividend;
dshifted = shift_left_n(d); // shift divisor left n bits
for(i = n - 1 ; 0 <= i ; i --)
{ if (remainder >= dshifted) { quotient = 1; remainder -= remainder; }
 dshifted = shift_right(dshifted) // shift_right function shifts value right one bit
}
```

# Signed division

- ▶ Easiest solution is to remember the sign of the operands and adjust the sign of the quotient and remainder accordingly
- ▶ A little problem:

$5 \div 2 = 2$  and the remainder is 1

$-5 \div 2 = -2$  and the remainder is -1

The sign of the remainder must match the sign of the quotient



# Floating point numbers

- ▶ Used to represent **real numbers**
- ▶ Very similar to **scientific notation**  
 $3.5 \times 10^6, 0.82 \times 10^{-5}, 75 \times 10^6, \dots$
- ▶ Both decimal numbers in scientific notation and floating point numbers can be normalized:  
 $3.5 \times 10^6, 8.2 \times 10^{-6}, 7.5 \times 10^7, \dots$



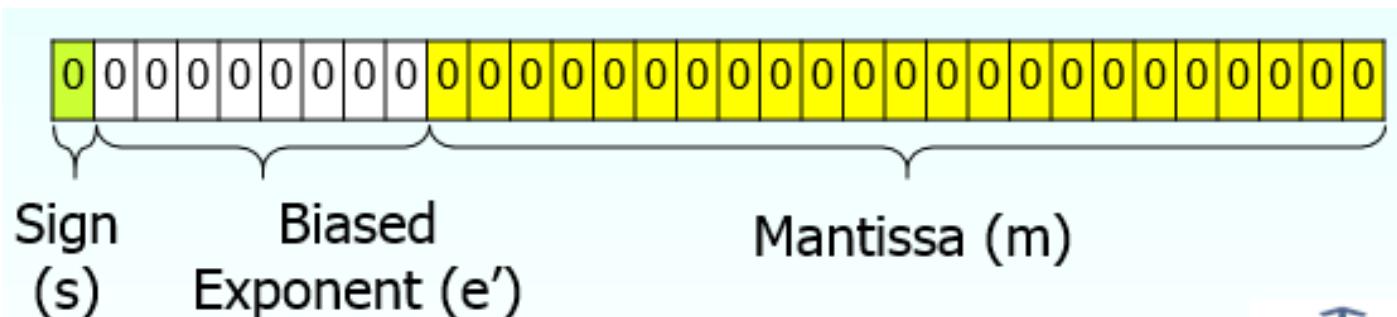
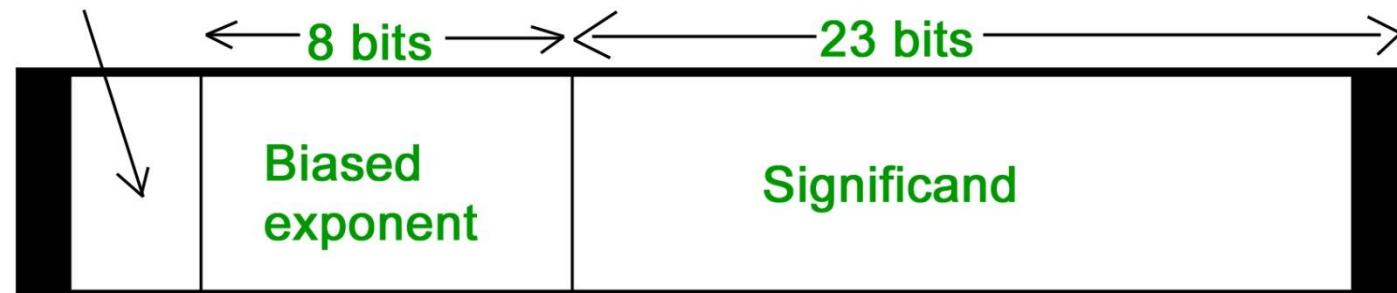
# Normalization

- ▶ Floating point numbers are usually normalized
- ▶ Exponent is adjusted so that leading bit (MSB) of mantissa is 1
- ▶ Since it is always 1 there is no need to store it
- ▶ Scientific notation where numbers are normalized to give a single digit before the decimal point like in. For example, we represent 3.625 in 32 bit format.
  - ▶ Changing 3 in binary=11
  - ▶ Changing 625 in binary= 101
  - ▶ Writing in binary exponent form
    - ▶  $3.625 = 11.101 \times 2^0$
  - ▶ On normalizing
    - ▶  $11.101 \times 2^0 = 1.1101 \times 2^1$

# 32 Bit Floating Point Representation of numbers

266

Sign of significand



► **For getting significand**

Digits after decimal = 1101

► Expanding to 23 bit = 1101000000000000000000000

► **Setting sign bit**

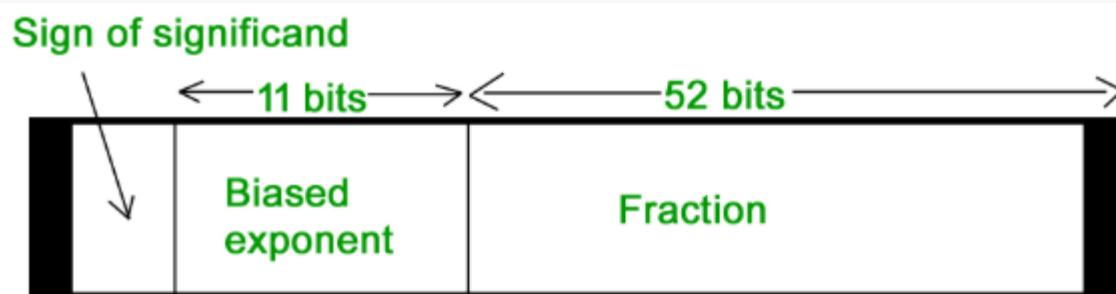
As it is a positive number, sign bit = 0

► Finally we arrange according to representation:

| Sign bit | exponent | significand               |
|----------|----------|---------------------------|
| 0        | 10000000 | 1101000000000000000000000 |



# 64-bit representation floating point numbers IEEE standard



Again we follow the same procedure upto normalization. After that, we add 1023 to bias the exponent.

For example, we represent -3.625 in 64 bit format.

Changing 3 in binary = 11

Changing .625 in binary= 1010

Writing in binary exponent form

- $3.625 = 11.101 \times 2^0$

On normalizing

- $11.101 \times 2^0 = 1.1101 \times 2^1$

On biasing exponent  $1023 + 1 = 1024$

- $(1024)_{10} = (10000000000)_2$

So 11 bit exponent = 10000000000

52 bit significand = 110100000000 ..... making total 52 bits

Setting sign bit = 1 (number is neg)

|   |                         |                                                                                  |
|---|-------------------------|----------------------------------------------------------------------------------|
| 1 | <b>10000000<br/>000</b> | <b>110100000000<br/>..... making total<br/>52 bits by adding<br/>further 0's</b> |
|---|-------------------------|----------------------------------------------------------------------------------|

- ▶ A recursive algorithm is an algorithm which calls itself with "smaller (or simpler)" input values, and which obtains the result for the current input by applying simple operations to the returned value for the smaller (or simpler) input.
- ▶ The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.
- ▶ Using recursive algorithm, certain problems can be solved quite easily.

## Properties

A recursive function can go infinite like a loop. To avoid infinite running of recursive function, there are two properties that a recursive function must have:

# Basic steps of recursive programs

Every recursive program follows the same basic sequence of steps:

1. **Initialize the algorithm.** Recursive programs often need a seed value to start with. This is accomplished either by using a parameter passed to the function or by providing a gateway function that is nonrecursive but that sets up the seed values for the recursive calculation.
2. **Check** to see whether the current value(s) being processed match the base case. If so, process and return the value.
3. **Redefine** the answer in terms of a smaller or simpler sub-problem or sub-problems.
4. **Run** the algorithm on the sub-problem.
5. **Combine the results** in the formulation of the answer.
6. **Return** the results.

## Example: Algorithm to find factorial number

**Fact(n)**

Begin

if n == 0 or 1 then

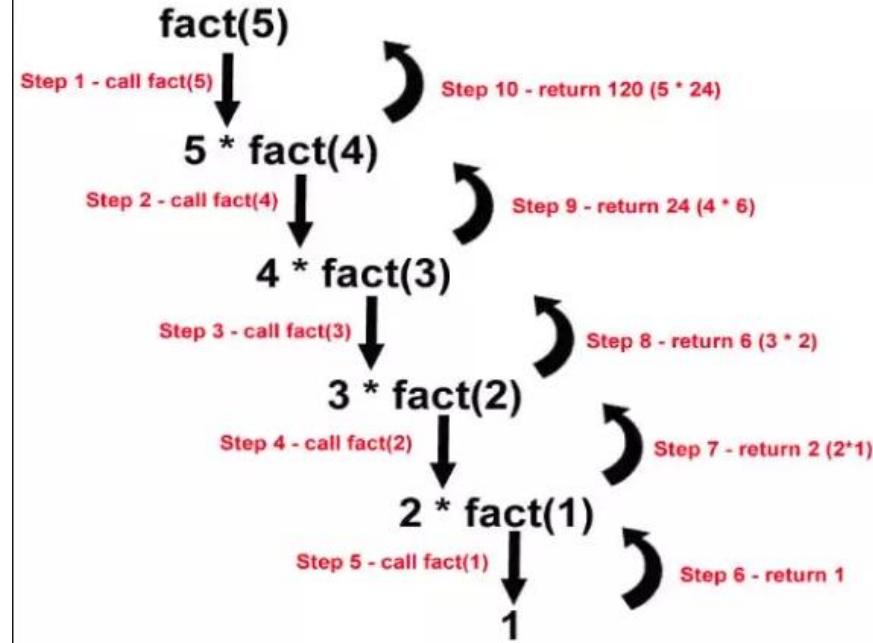
    Return 1;

else

    Return n\*Call Fact(n-1);

endif

End

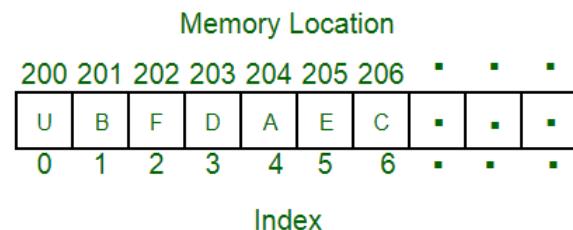


# Data Structure

- ▶ A data structure is a particular way of organizing data in a computer so that it can be used effectively.
- ▶ The idea is to reduce the space and time complexities of different tasks.
- ▶ For example, we can store a list of items having the same data-type using the array data structure.

- ▶ Some popular Linear data structure:

1. Array
2. Linked List
3. Stack
4. Queue

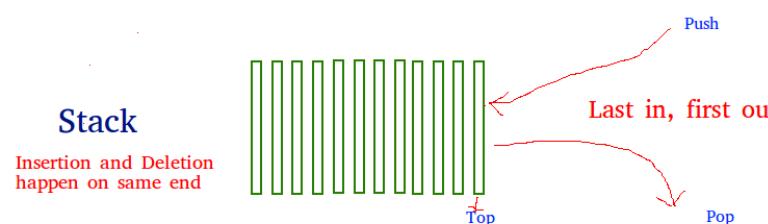


# STACK

274



- ▶ A stack is an Abstract Data Type (ADT), commonly used in most programming languages.
- ▶ It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.
- ▶ A real-world stack allows operations at one end only.
- ▶ For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.
- ▶ This feature makes stack a Last-in-first-out (LIFO) or a First-in-last-out (FILO) data structure.

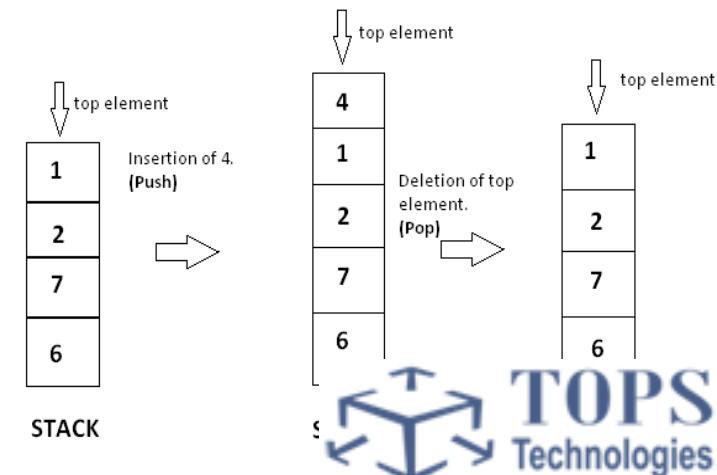


stands for Last-in-first-out or a  
TOPS Technologies

# STACK

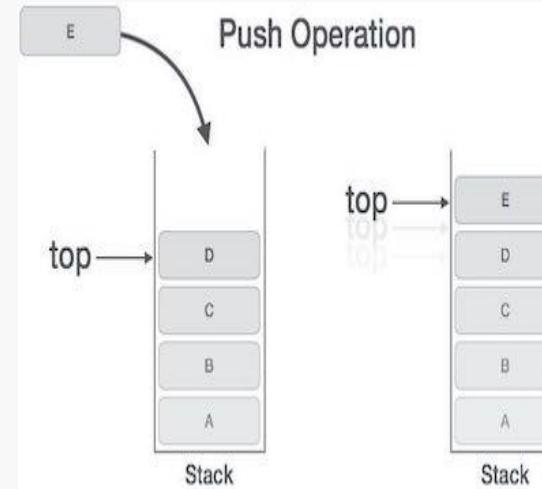
Mainly the following basic operations are performed in the stack:

- ▶ **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- ▶ **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- ▶ **Peek or Top:** Returns top element of stack.
- ▶ **isEmpty:** Returns true if stack is empty, else false.



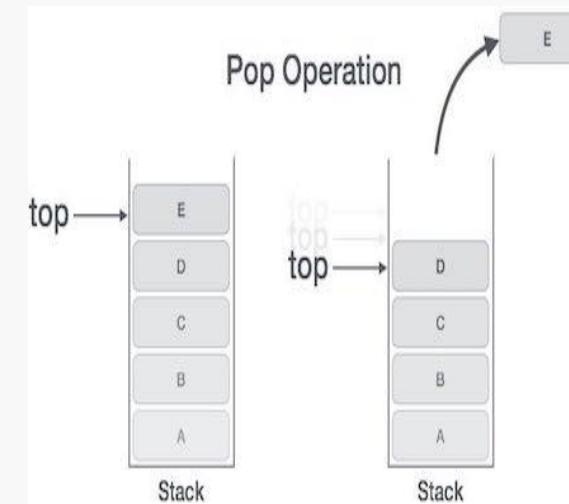
## 1. Push Operation Algorithm:

- 1) IF TOP = MAX then  
Print "Stack is full";  
Exit;
- 2) Otherwise  
TOP:=TOP + 1; /\*increment TOP\*/  
STACK (TOP):= ITEM;
- 3) End of IF
- 4) Exit



## 2. Pop Operation Algorithm:

- 1) IF TOP = 0 then  
Print "Stack is empty";  
Exit;
- 2) Otherwise  
ITEM:=STACK (TOP);  
TOP:=TOP - 1;
- 3) End of IF
- 4) Exit



# STACK

There are two ways to implement a stack:

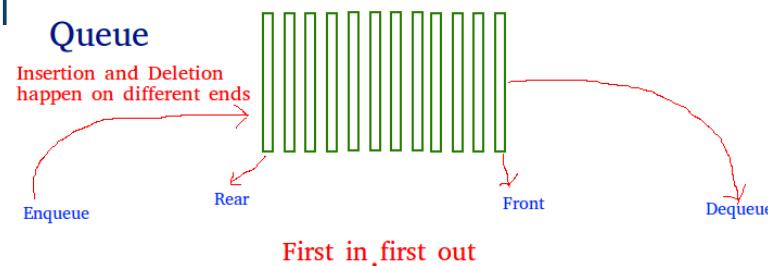
- ▶ Using array
- ▶ Using linked list
  
- ▶ **Refer This Example:**
- ▶ **Stack Using Array:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Stack/stackArray.cpp>
  
- ▶ **Stack using Linked List:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Stack/stackLinked>

# QUEUE



278

- ▶ A Queue is a linear structure which follows a particular order in which the operations are performed.
- ▶ The order is First In First Out (FIFO).
- ▶ A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first.
- ▶ The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item t

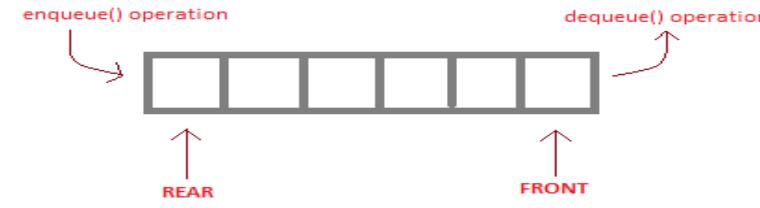


# Queue Operations

There are five operations possible on a queue:

- ▶ Initialize operation
- ▶ **Enqueue** :Addition or insertion of element.
- ▶ **Dequeue**: Deletion operation.
- ▶ **Is\_full check.**
- ▶ **Is\_empty check.**

Two pointers **REAR** and **FRONT** are maintained by Queue for Enqueue and Dequeue



`enqueue()` is the operation for adding an element into Queue.

`dequeue()` is the operation for removing an element from Queue .

## QUEUE DATA STRUCTURE



## Enqueue Algorithm:

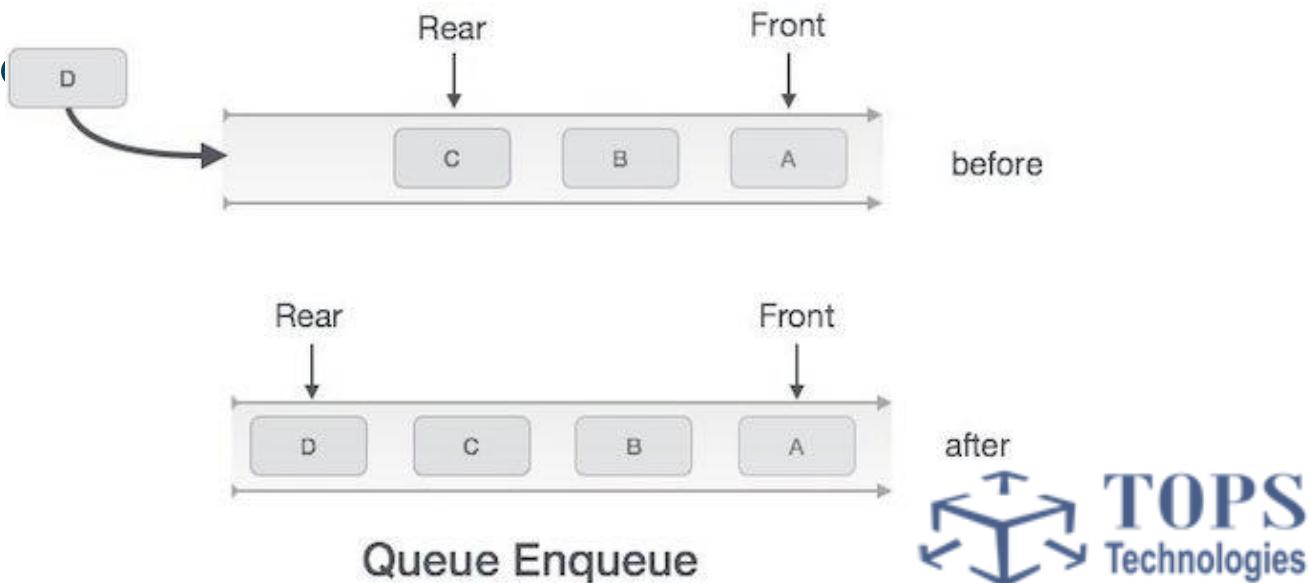
Step 1 – Check if the queue is full.

Step 2 – If the queue is full, produce overflow error and exit.

Step 3 – If the queue is not full, increment rear pointer to point the next empty space.

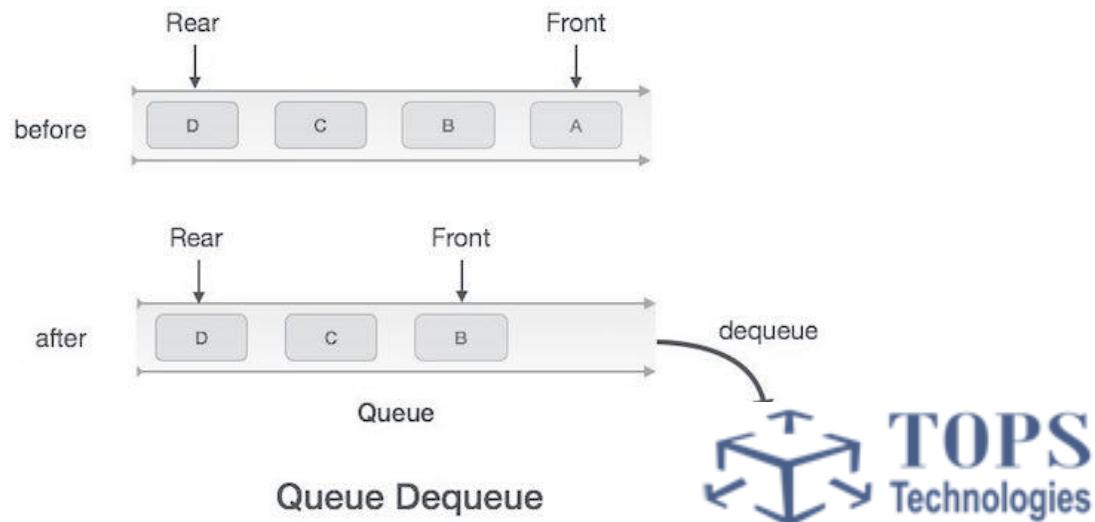
Step 4 – Add data element to the queue location, where the rear is pointing.

Step 5 – return success



## Dequeue Algorithm

- Step 1 – Check if the queue is empty.
- Step 2 – If the queue is empty, produce underflow error and exit.
- Step 3 – If the queue is not empty, access the data where front is pointing.
- Step 4 – Increment front pointer to point to the next available data element.
- Step 5 – Return success.



# Refer This Example

- ▶ **Queue using Array:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Queue/queueArray.cpp>
  
- ▶ **Queue using Linked List:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Queue/queueLinked List.cpp>

# QUEUE VARIATIONS

The standard queue data structure has the following variations:

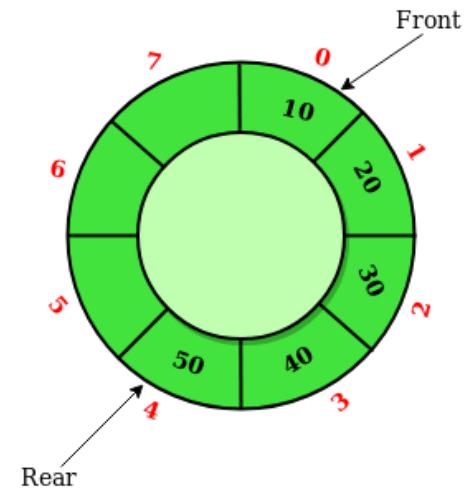
- ▶ Double-ended queue
- ▶ Circular queue

## Double-ended queue

- ▶ In a standard queue, a character is inserted at the back and deleted in the front. However, in a double-ended queue, characters can be inserted and deleted from both the front and back of the queue.

## Circular Queue:

- ▶ in a circular queue, vacant spaces are reutilized.
- ▶ While inserting elements, when you reach the end of an array and you need to insert another element, you must insert that element at the beginning (given that the first element has been deleted and the space is vacant).



# Module 3 Topics:

- DBMS and RDBMS
- E-R Relational Schema
- Types of database
- Normalization
- Algebra
- Database Programming language SQL
- Keys: Primary Key, Unique Key, Foreign Key
- SQL Statement types: DML, DDL, TQL, TCL
- Joins
- Function
- Procedure
- Trigger
- Transaction Concept (Properties, Rollback, Commit, Save Point)
- Cursor
- Database Backup and Recovery

- ▶ DBMS stands for **Data Base Management System**.

Data + Management System

- ▶ **Database** is a collection of inter-related data and Management System is a set of programs to store and retrieve those data.
- ▶ **DBMS** is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.
- ▶ For Example, university database organizes the data of students, faculty, and admin staff etc. which helps in efficient retrieval, insertion and deletion of data from it



- A DBMS consists of 2 main pieces:

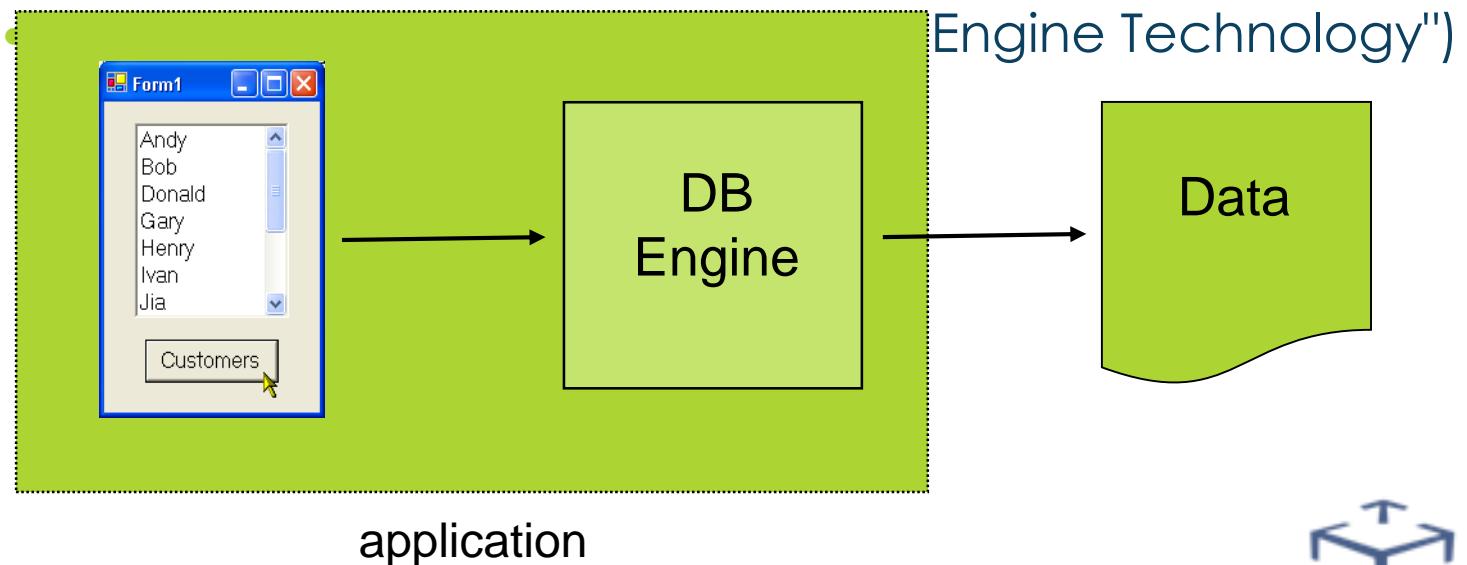
- the data
- the DB engine
- the data is typically stored in files



- Two most common types of DBMS are:
  - Local
  - Server

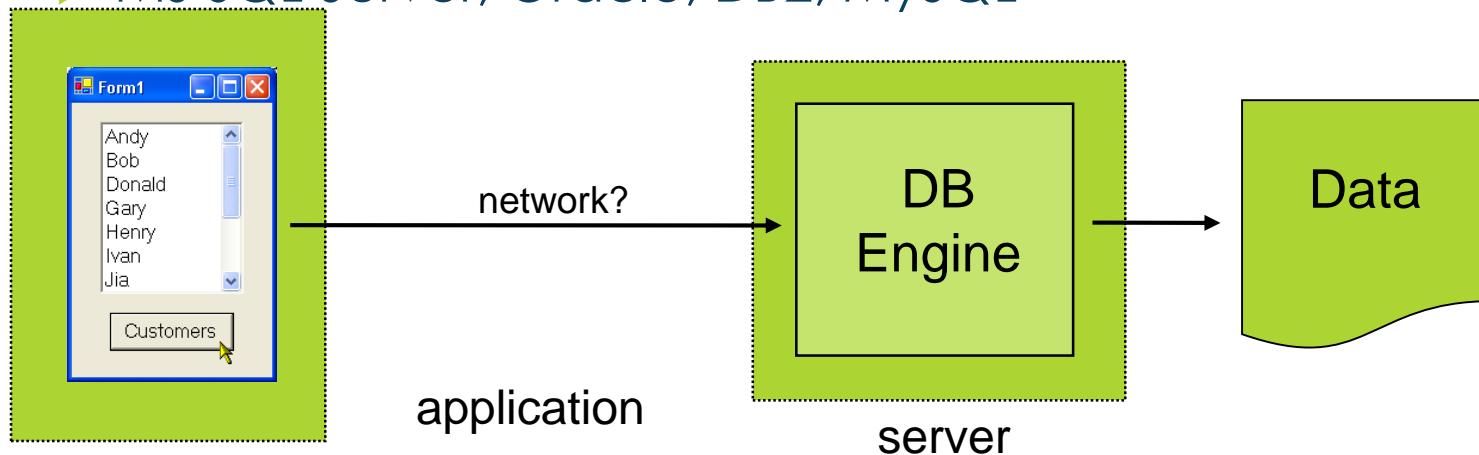
# Local DBMS

- A local DBMS is where DB engine runs as part of application
- Example?
  - MS Access



# Server DBMS

- ▶ A server DBMS is where DB engine runs as a separate process
  - ▶ typically on a different machine (i.e. server)
- ▶ Examples?
  - ▶ MS SQL Server, Oracle, DB2, MySQL



# Popular DBMS Software

Here, is the list of some popular DBMS system:

- MySQL
- Microsoft Access
- Oracle
- PostgreSQL
- dBASE
- FoxPro
- SQLite
- IBM DB2
- LibreOffice Base
- MariaDB
- Microsoft SQL Server etc.

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: Storage of data and retrieval of data.

## Storage:

- ▶ According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage.

**Fast Retrieval of data:** Along with storing the data in an optimized and systematic manner, it is also important that we retrieve it quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.



# PURPOSE OF DBMS

- ▶ The main purpose of database systems is to manage the data.

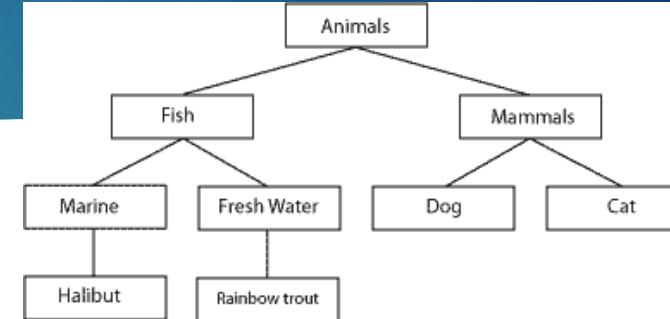
Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.



- ▶ Stands for "Relational Database Management System."
- ▶ An RDBMS is a type of DBMS designed specifically for relational databases.
- ▶ A relational database refers to a database that stores data in a structured format, using rows and columns.
- ▶ This makes it easy to locate and access specific values within the database.
- ▶ It is "relational" because the values within each table are related to each other.
- ▶ Tables may also be related to other tables. The relational structure makes it possible to run queries across multiple tables at once.
- ▶ The RDBMS refers to the that executes queries on the data, including adding, updating, and software searching for values.
- ▶ An RDBMS may also provide a visual representation of

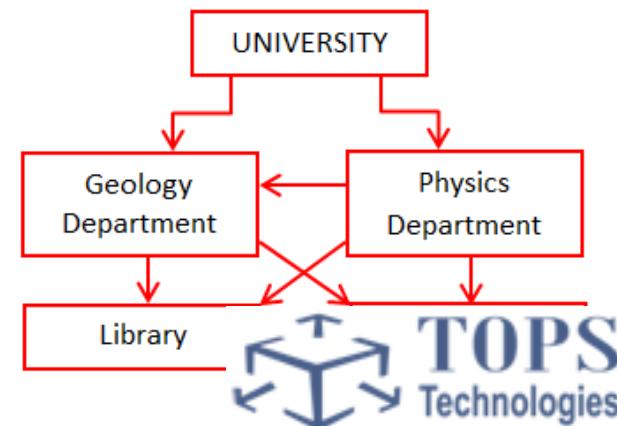
## 1. Hierarchical databases -

- ▶ It is very fast and simple.
- ▶ This kind of database model uses a tree-like structure which links a number of dissimilar elements to one primary record – the "owner" or "parent".
- ▶ Each record in a hierarchical database contains information about a group of parent child relationships.



## 2. Network databases –

- ▶ The network database model can be viewed as a net-like form where a single element can point to multiple data elements and can itself be pointed to by multiple data elements.



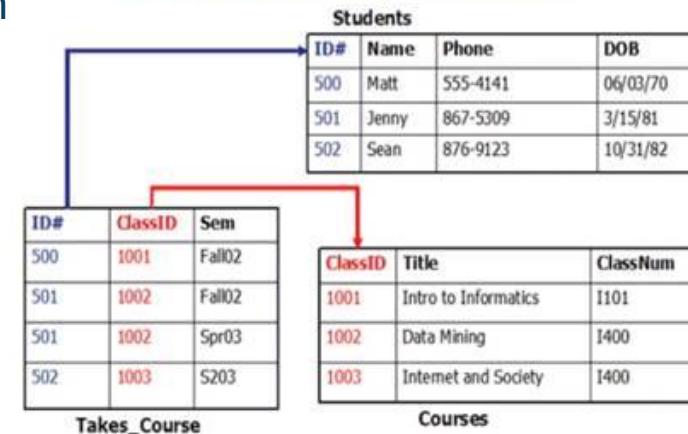
### 3. Relational databases –

- A relational database is one in which data is stored in the form of tables, using rows and columns.
- This arrangement makes it easy to locate and access specific data within the database. It is “relational” because the data within each table are related to each other.

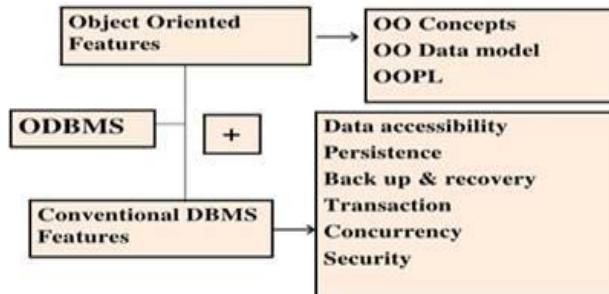
### 4. Object-oriented databases -

- The recent development in database technology is the incorporation of the object concept that has become significant in programming languages.
- In object-oriented databases, all data are objects. Objects may be linked to each other by an “is-part-of” relationship to represent larger, composite objects
- Object DBMS's increase the semantics of the C++ and Java

## Relational DBMS



## ODBMS



# Relational Databases

- ▶ RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- ▶ Most of today's databases are relational:
  - ▶ database contains 1 or more tables
  - ▶ table contains 1 or more records
  - ▶ record contains 1 or more fields
  - ▶ fields contain the data
- ▶ So why is it called "relational"?



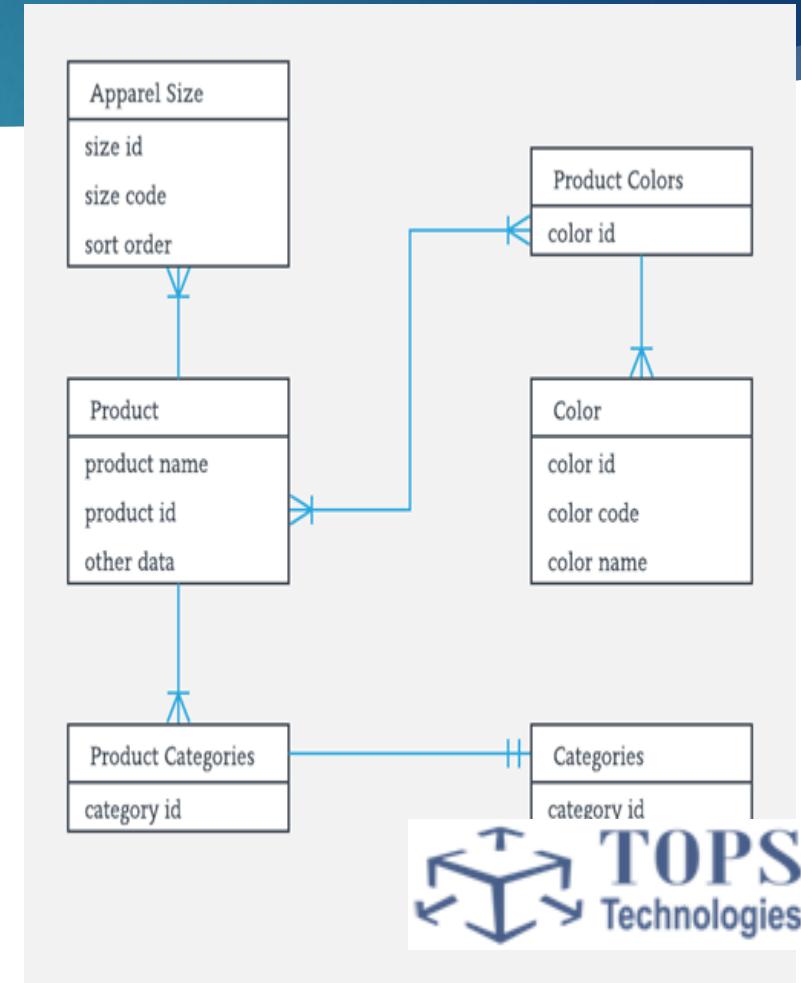
# E-R Model

- ▶ The ER or (Entity Relational Model) is a high-level conceptual data model diagram.
- ▶ Entity-Relation model is based on the notion of real-world entities and the relationship between them.
- ▶ ER modeling helps you to analyze data requirements systematically to produce a well-designed database.
- ▶ So, it is considered a best practice to complete ER modeling before implementing your database.



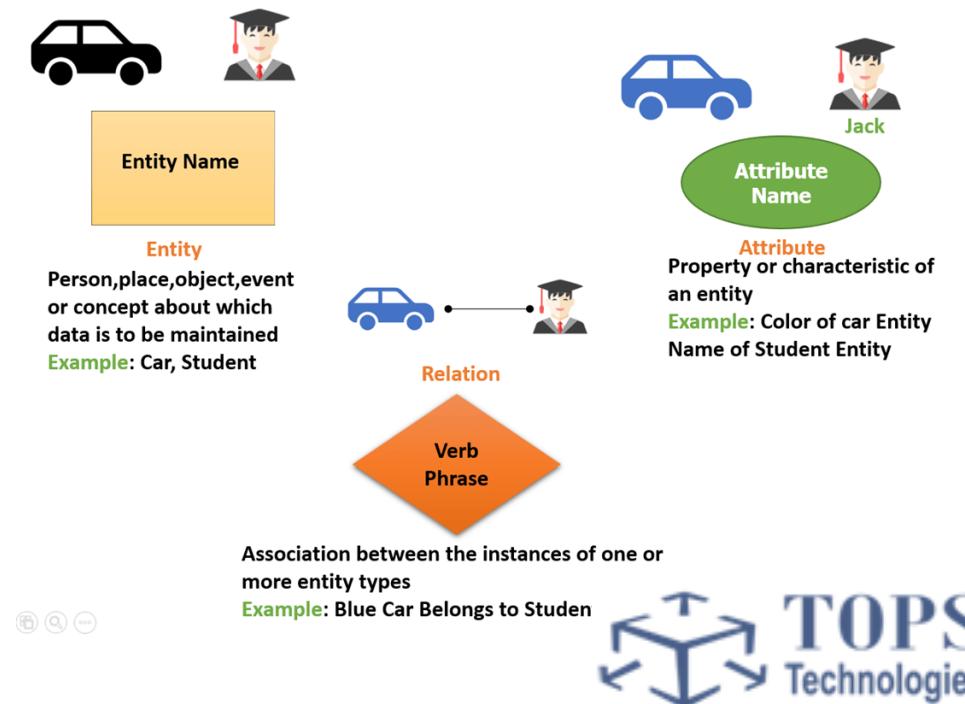
# ER Diagram

- ▶ Entity relationship diagram displays the relationships of entity set stored in a database.
- ▶ In other words, we can say that ER diagrams help you to explain the logical structure of databases.
- ▶ At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.



# Components of ER Diagram

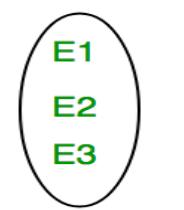
- ▶ Entities
- ▶ Attributes
- ▶ Relationships
  
- ▶ For example, in a University database, we might have entities for Students, Courses, and Lecturers. Students entity can have attributes like Rollno, Name, and DeptID. They might have relationships with Courses and Lecturers.



- ▶ A real-world thing either living or non-living that is easily recognizable and non recognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world.
- ▶ An entity can be place, person, object, event or a concept, which stores data in the database. Entities are must have an attribute, and made up of some 'attributes' which are characteristics of entities.
- ▶ Set of all entity is called **entity set**.



Characteristics of entities  
Every entity is  
that entity.



Examples of entities:

- ▶ **Person:** Employee, Student, Patient

# Attribute

Attributes are the properties which define the entity type. For example, Roll\_No, Name, DOB, Age, Address, Mobile\_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

Attribute

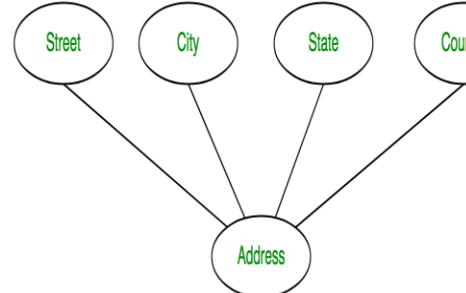
Roll\_No

## 1. Key Attribute:

- ▶ The attribute which uniquely identifies each entity in the entity set is called key attribute.
- ▶ For example, Roll\_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

## 2. Composite Attribute –

An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals



## 3. Multivalued Attribute –

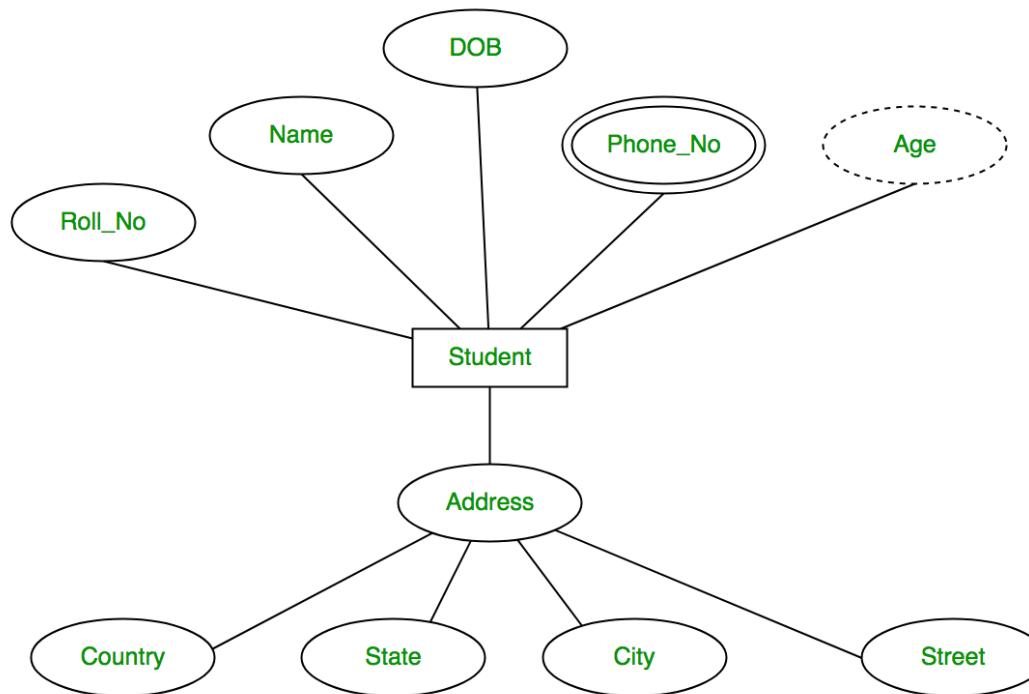
An attribute consisting more than one value for a given entity. For example, Phone\_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.



## 4. Derived Attribute –

An attribute which can be derived from other attributes of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

The complete entity type Student with its attributes can be represented as:

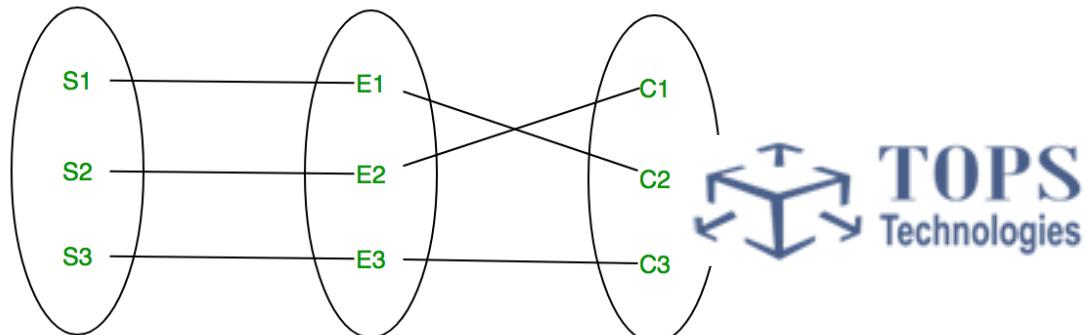


# Relationship Type and Relationship Set:

- A relationship type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity types Student and Course. A relationship type is represented by a diamond shape connecting the entities with lines.



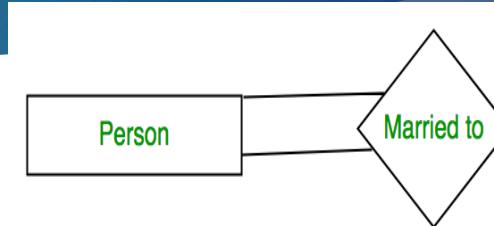
A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



# Degree of a relationship set:

## 1. Unary Relationship –

When there is only ONE entity set participating in a relation, the relationship is called as unary relationship. For example, one person is married to only one person.



## 2. Binary Relationship –

When there are TWO entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course.



## 3. n-ary Relationship –

When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

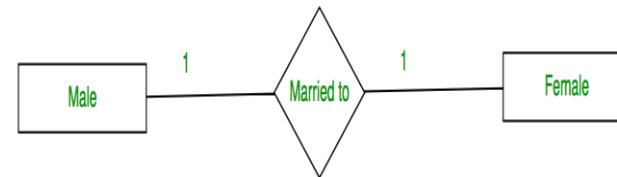


# Cardinality:

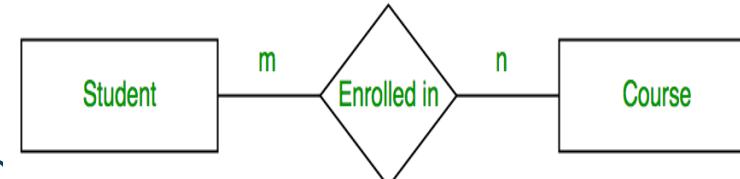
The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

- ▶ **One to one** – When each entity in each entity set can take part only once in the relationship, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one

**Many to one** – When one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one.



**Many to many** – When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.



### Participation Constraint:

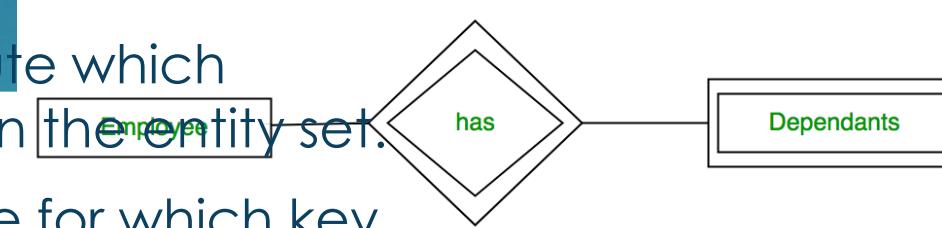
Participation Constraint is applied on the entity participating in the relationship set.

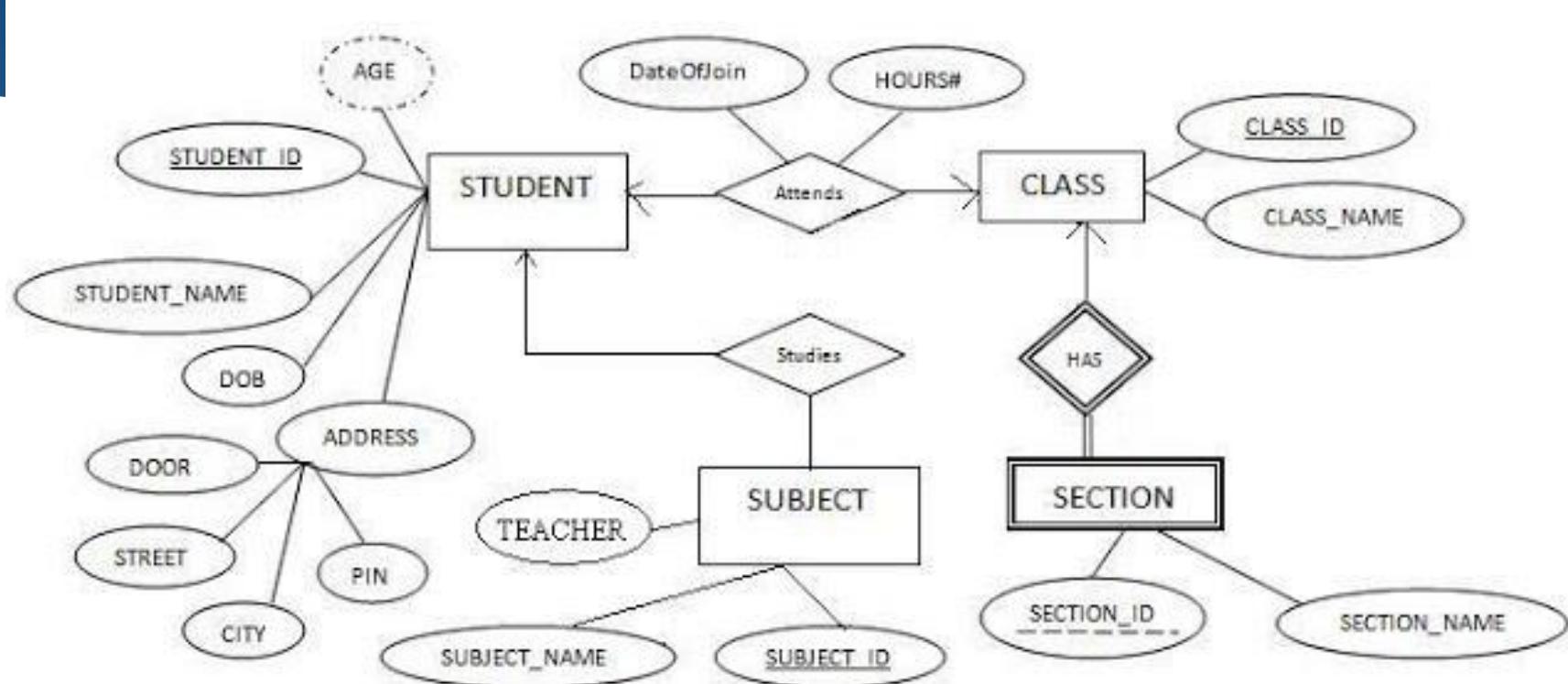
**Total Participation** – Each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

**Partial Participation** – The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial. The diagram depicts the 'Enrolled in' relationship set with Student Entity having total participation and Course Entity set having partial participation.

## ► Weak Entity Type and Identifying Relationship:

- An entity type has a key attribute which uniquely identifies each entity in the entity set.
- But there exists some entity type for which key attribute can't be defined. These are called Weak Entity type.
- For example, A company may store the information of dependants (Parents, Children, Spouse) of an Employee. But the dependants don't have existence without the employee. So Dependant will be weak entity type and Employee will be Identifying Entity type for Dependant.





- Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

## **Unary Relational Operations Operations**

- SELECT (symbol:  $\sigma$ )
- PROJECT (symbol:  $\pi$ )
- RENAME (symbol: )

## **Binary Relational Operations**

- JOIN
- DIVISION

## **Relational Algebra Operations From Set Theory**

- UNION ( $\cup$ )
- INTERSECTION ( ),

- DIFFERENCE (-)



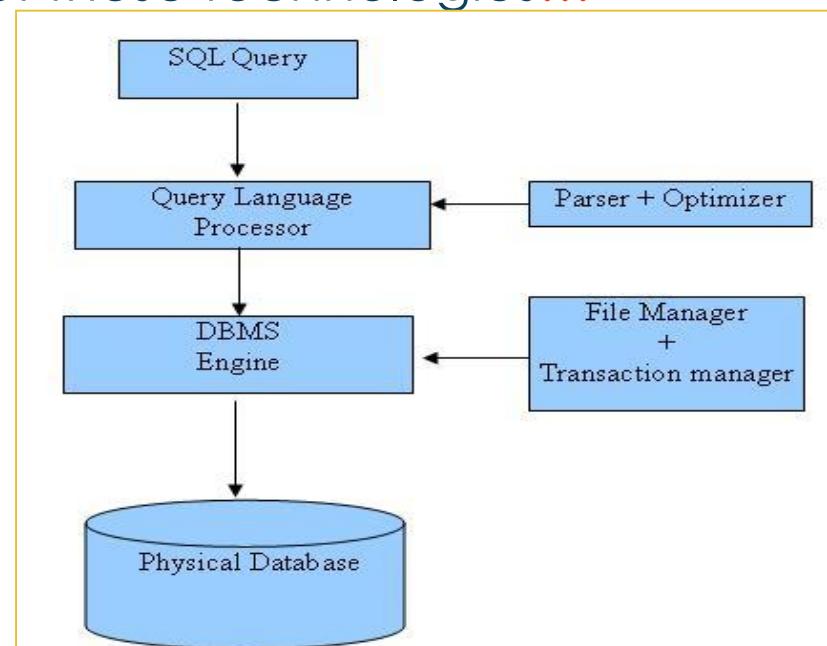
# What is SQL?

- ▶ SQL is **Structured Query Language**, which is a computer language for **storing, manipulating and retrieving data** stored in relational database.
- ▶ SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.
- ▶ SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.
- ▶ Also, they are using different dialects, such as:
  - ▶ MS SQL Server using T-SQL, ANSI SQL
  - ▶ Oracle using PL/SQL,
  - ▶ MS Access version of SQL is called JET SQL (native format)



# Objectives

“The large majority of today's business applications revolve around relational databases and the SQL programming language (Structured Query Language). Few businesses could function without these technologies...”



# Why SQL?

- ▶ Allows users to access data in relational database management systems.
- ▶ Allows users to describe the data.
- ▶ Allows users to define the data in database and manipulate that data.
- ▶ Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- ▶ Allows users to create and drop databases and tables.
- ▶ Allows users to create view, stored procedure, functions in a database.
- ▶ Allows users to set permissions on tables, procedure, views



# What is SQL? (Cont...)

- SQL stands for Structured Query Language
- SQL allows you to access a database
- SQL is an ANSI standard computer language
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn
- SQL is written in the form of queries
- action queries insert, update & delete data
- select queries retrieve data from DB

## ► Primary Key:

- ▶ A primary key is a column of table which uniquely identifies each tuple (row) in that table.
- ▶ Primary key enforces integrity constraints to the table.
- ▶ Only one primary key is allowed to use in a table.
- ▶ The primary key does not accept the any duplicate and NULL values.
- ▶ The primary key value in a table changes very rarely so it is chosen 'seldom'.
- ▶ A primary key of a

| Table : Student |      |         |              |            |
|-----------------|------|---------|--------------|------------|
| Roll_number     | Name | Batch   | Phone_number | Citizen_ID |
| 101             | John | Batch-A | 9876543210   | 1234567890 |

Primary key



## ► Unique Key:

- Unique key constraints also identifies an individual table uniquely in a relation or table.
- A table can have more than one unique key unlike primary key.
- Unique key constraints can accept only one NULL value for column.
- Unique constraints are also referenced by the foreign key of another tab
- It can be used as constraints compared to primary key.

Table : Student

| Roll_number | Name | Batch   | Phone_number | Citizen_ID |
|-------------|------|---------|--------------|------------|
| 12345       | John | Batch A | 9876543210   | 1234567890 |

Unique key



## ► Foreign Key:

- When, "one" table's primary key field is added to a related "many" table in order to create the common field which relates the two tables, it is called a foreign key in the "many" table.
- In the example given below, salary of an employee is stored in salary table. Relation is established via is stored in "Employee" table. To identify the salary of "Jforeign key column "Employee\_ID\_Ref" which refers "Employee\_ID" field in Employee table.

| Table : Employee |               | Table : Salary  |      |       |        |
|------------------|---------------|-----------------|------|-------|--------|
| Employee_ID      | Employee_Name | Employee_ID_Ref | Year | Month | Salary |
| 1                | Jhon          | 1               | 2012 | April | 30000  |
| 2                | Alex          | 1               | 2012 | May   | 31000  |
| 3                | James         | 1               | 2012 | June  | 32000  |
| 4                | Roy           | 2               | 2012 | April | 40000  |
| 5                | Kay           | 2               | 2012 | May   | 41000  |
|                  |               | 2               | 2012 | June  | 42000  |

Employee info  
stored with each



- ▶ Normalization is the process of minimizing redundancy (duplicity) from a relation or set of relations.
- ▶ Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations.

## Most Commonly used normal forms:

- ▶ First normal form(1NF)
- ▶ Second normal form(2NF)
- ▶ Third normal form(3NF)
- ▶ Boyce & Codd normal form (BCNF)

# First Normal Form

- ▶ If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute.
- ▶ A relation is in first normal form if every attribute in that relation is singled valued attribute.

Example 1 – Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD\_PHONE. Its decomposition into 1NF has been

| STUD_NO | STUD_NAME | STUD_PHONE                | STUD_STATE | STUD_COUNTRY |
|---------|-----------|---------------------------|------------|--------------|
| 1       | RAM       | 9716271721,<br>9871717178 | HARYANA    | INDIA        |
| 2       | RAM       | 9898297281                | PUNJAB     | INDIA        |
| 3       | SURESH    |                           | PUNJAB     | INDIA        |

Table 1

Conversion to first normal form

| STUD_NO | STUD_NAME | STUD_PHONE | STUD_STATE | STUD_COUNTRY |
|---------|-----------|------------|------------|--------------|
| 1       | RAM       | 9716271721 | HARYANA    |              |
| 1       | RAM       | 9871717178 | HARYANA    | INDIA        |
| 2       | RAM       | 9898297281 | PUNJAB     | INDIA        |
| 3       | SURESH    |            | PUNJAB     | INDIA        |

Table 2

# Second Normal Form

- ▶ To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency.
- ▶ relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
- ▶ Partial Dependency – If the proper subset of candidate key determines non-prime dependency.
- ▶ The table is not in 2nf for

| STUD_NO | COURSE_NO | COURSE_FEE |
|---------|-----------|------------|
| 1       | C1        | 1000       |
| 2       | C2        | 1500       |
| 1       | C4        | 2000       |
| 4       | C3        | 1000       |
| 4       | C1        | 1000       |
| 2       | C5        | 2000       |

- Note that, there are many courses having the same course fee.
- COURSE\_FEE cannot alone decide the value of COURSE\_NO or STUD\_NO;
- COURSE\_FEE together with STUD\_NO cannot decide the value of COURSE\_NO;
- COURSE\_FEE together with COURSE\_NO cannot decide the value of STUD\_NO;
- Hence, COURSE\_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD\_NO, COURSE\_NO} ;
- But, COURSE\_NO  $\rightarrow$  COURSE\_FEE , i.e., COURSE\_FEE is dependent on COURSE\_NO, which is a proper subset of the candidate key.
- Non-prime attribute COURSE\_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,  
we need to split the table into two tables such as :  
Table 1: STUD\_NO, COURSE\_NO  
Table 2: COURSE\_NO, COURSE\_FEE

| STUD_NO | COURSE_NO | COURSE_FEE |
|---------|-----------|------------|
| 1       | C1        | 1000       |
| 2       | C2        | 1500       |
| 1       | C4        | 2000       |
| 4       | C3        | 1000       |
| 4       | C1        | 1000       |
| 2       | C5        | 2000       |



| Table 1 |           | Table 2   |            |
|---------|-----------|-----------|------------|
| STUD_NO | COURSE_NO | COURSE_NO | COURSE_FEE |
| 1       | C1        | C1        | 1000       |
| 2       | C2        | C2        | 1500       |
| 1       | C4        | C3        | 1000       |
| 4       | C3        | C4        | 2000       |
| 4       | C1        | C5        | 2000       |

# Third Normal Form

- ▶ A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.
- A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency  $X \rightarrow Y$ 
  - ▶ X is a super key.
  - ▶ Y is a prime attribute (each element of Y is part of some candidate key).

| STUD_NO | STUD_NAME | STUD_STATE | STUD_COUNTRY | STUD_AGE |
|---------|-----------|------------|--------------|----------|
| 1       | RAM       | HARYANA    | INDIA        | 20       |
| 2       | RAM       | PUNJAB     | INDIA        | 19       |
| 3       | SURESH    | PUNJAB     | INDIA        | 21       |

Table 4



In relation STUDENT given in Table 4, FD set: {STUD\_NO  $\rightarrow$  STUD\_NAME, STUD\_NO  $\rightarrow$  STUD\_STATE, STUD\_STATE  $\rightarrow$  STUD\_COUNTRY, STUD\_NO  $\rightarrow$  STUD\_AGE}  
Candidate Key: {STUD\_NO}

For this relation in table 4, STUD\_NO  $\rightarrow$  STUD\_STATE and STUD\_STATE  $\rightarrow$  STUD\_COUNTRY are true. STUD\_COUNTRY is transitively dependent on STUD\_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY, STUD\_AGE) as:  
STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_AGE)  
STATE\_COUNTRY (STATE, COUNTRY)

# Boyce Codd normal form (BCNF)

- ▶ It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF.
- ▶ A table complies with BCNF if it is in 3NF and for every functional dependency  $X \rightarrow Y$ , X should be the super key of the table.

**Example:** Suppose there is a company wherein employees work in more than one department. They store t

- ▶ **Functional dependencies :**  
 $\text{emp\_id} \rightarrow \text{emp\_nationality}$   
 $\text{emp\_dept} \rightarrow \{\text{dept\_type}, \text{dept\_no\_of\_e}$
- ▶ **Candidate key:**  $\{\text{emp\_id}, \text{emp\_dept}\}$   
The table is not in BCNF as neither  $\text{emp\_id}$  nor  $\text{emp\_dept}$  alone are keys.

| emp_id | emp_nationality | emp_dept                     | dept_type | dept_no_of_emp |
|--------|-----------------|------------------------------|-----------|----------------|
| 1001   | Austrian        | Production and planning      | D001      | 200            |
| 1001   | Austrian        | stores                       | D001      | 250            |
| 1002   | American        | design and technical support | D134      | 100            |
| 1002   | American        | Purchasing department        | D134      | 600            |

- To make the table comply with BCNF we can break the table in three tables like this:

**emp\_nationality table:**

| emp_id | emp_nationality |
|--------|-----------------|
| 1001   | Austrian        |
| 1002   | American        |

**emp\_dept table:**

| emp_dept                     | dept_type | dept_no_of_emp |
|------------------------------|-----------|----------------|
| Production and planning      | D001      | 200            |
| stores                       | D001      | 250            |
| design and technical support | D134      | 100            |
| Purchasing department        | D134      | 600            |

**emp\_dept\_mapping table:**

| emp_id | emp_dept                     |
|--------|------------------------------|
| 1001   | Production and planning      |
| 1001   | stores                       |
| 1002   | design and technical support |
| 1002   | Purchasing department        |

- ▶ When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.
- ▶ There are various components included in the process.
  - ▶ These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc.
  - ▶ Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

# SQL Statement Types

- ▶ **DDL** – Data Definition Language
- ▶ **DML** – Data Manipulation Language
- ▶ **DCL** – Data Control Language
- ▶ **DQL** – Data Query Language

## SQL Join Types

- **INNER JOIN**: returns rows when there is a match in both tables.
- **LEFT JOIN**: returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN**: returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN**: returns rows when there is a match in one of the tables.



# DDL - Data Definition Language

| Command | Description                                                                 |
|---------|-----------------------------------------------------------------------------|
| CREATE  | Creates a new table, a view of a table, or other object in database         |
| ALTER   | Modifies an existing database object, such as a table.                      |
| DROP    | Deletes an entire table, a view of a table or other object in the database. |

# DQL – Data Query Language

| Command | Description                                       |
|---------|---------------------------------------------------|
| SELECT  | Retrieves certain records from one or more tables |

# DML – Data Manipulation Language

| Command | Description      |
|---------|------------------|
| INSERT  | Creates a record |
| UPDATE  | Modifies records |
| DELETE  | Deletes records  |

# DCL – Data Control Language

| Command | Description                             |
|---------|-----------------------------------------|
| GRANT   | Gives a privilege to user               |
| REVOKE  | Takes back privileges granted from user |

# Create, Drop, Use Database Syntax

## ► **SQL CREATE DATABASE STATEMENT**

```
CREATE DATABASE database_name;
```

## ► **SQL DROP DATABASE Statement:**

```
DROP DATABASE database_name;
```

## ► **SQL USE STATEMENT**

```
USE DATABASE database_name;
```



## ► SQL CREATE TABLE STATEMENT

```
CREATE TABLE table_name(column1 datatype, column2 datatype,
column3 datatype, , columnN datatype, PRIMARY KEY(one or
more columns));
```

## ► SQL DROP TABLE STATEMENT

```
DROP TABLE table_name;
```

## ► SQL TRUNCATE TABLE STATEMENT

```
TRUNCATE TABLE table_name;
```

## ► SQL ALTER TABLE STATEMENT

```
ALTER TABLE
table_name{ADD | DROP | MODIFY}column_name{data_type}
```



## ► SQL ALTER TABLE STATEMENT (RENAME)

## ► SQL INSERT INTO STATEMENT

```
INSERT INTO table_name(column1, column2....columnN) VALUES (
value1, value2....valueN);
```

## ► SQL UPDATE STATEMENT

```
UPDATE table_name SET column1 = value1, column2 =
value2....columnN=valueN [WHERE CONDITION];
```

## ► SQL DELETE STATEMENT

```
DELETE FROM table_name WHERE {CONDITION};
```



# Select Statement Syntax

## ► SQL SELECT STATEMENT

```
SELECT column1, column2....columnN FROM table_name;
```

## ► SQL DISTINCT CLAUSE

```
SELECT DISTINCT column1, column2....columnN FROM table_name;
```

## ► SQL WHERE CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE
CONDITION;
```

## ► SQL AND/OR CLAUSE

```
SELECT column1, column2....columnN FROM table_name
CONDITION-1 {AND | OR} CONDITION-2;
```

# Select Statement Syntax

## ► SQL IN CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE
column_name IN (val-1, val-2,...val-N);
```

## ► SQL BETWEEN CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE
column_name BETWEEN val-1 AND val-2;
```

## ► SQL LIKE CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE
column_name LIKE { PATTERN };
```

## ► SQL ORDER BY CLAUSE

```
SELECT column1, column2....columnN FROM table_name
TOPS TECHNOLOGIES PVT. LTD. SEP 2019
CONDITION ORDER BY column_name {ASC | DESC};
```

# Select Statement Syntax

## ► SQL GROUP BY CLAUSE

```
SELECT SUM(column_name) FROM table_name WHERE CONDITION
GROUP BY column_name;
```

## ► SQL COUNT CLAUSE

```
SELECT COUNT(column_name)FROM table_name WHERE
CONDITION;
```

## ► SQL HAVING CLAUSE

```
SELECT SUM(column_name) FROM table_name WHERE
GROUP BY column_name HAVING (arithmeticfunction c
```



► **SQL CREATE INDEX Statement:**

```
CREATE UNIQUE INDEX index_name ON table_name(column1,
column2,...columnN);
```

► **SQL DROP INDEX STATEMENT**

```
ALTER TABLE table_name DROP INDEX index_name;
```

► **SQL DESC Statement :**

```
DESC table_name;
```



## ► SQL COMMIT STATEMENT

COMMIT;

## ► SQL ROLLBACK STATEMENT

ROLLBACK;



# JOIN

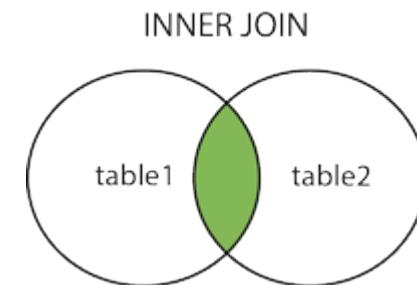
- ▶ A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.
- ▶ Different types of Joins are:
  1. INNER JOIN
  2. LEFT JOIN
  3. RIGHT JOIN
  4. FULL JOIN



# Inner Join Syntax

- ▶ The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN.
- ▶ The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. T
- ▶ he query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.
- ▶ SYNTAX:

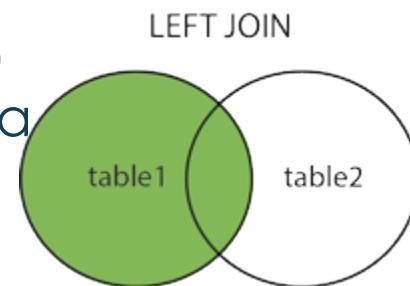
```
SELECT table1.column1, table2.column2...FROM
table1INNER JOIN table2ON table1.common_field =
table2.common_field;
```



# Left Join Syntax

- ▶ The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.
- ▶ This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.
- ▶ SYNTAX:

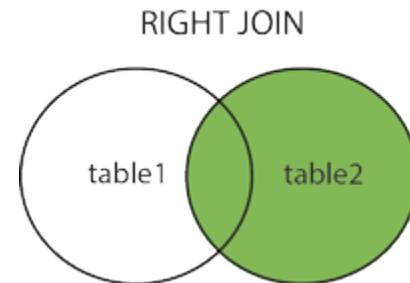
```
SELECT table1.column1, table2.column2...FROM
table1LEFT JOIN table2ON table1.common_field =
table2.common_field;
```



# Right Join Syntax

- ▶ The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.
- ▶ This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.
- ▶ SYNTAX:

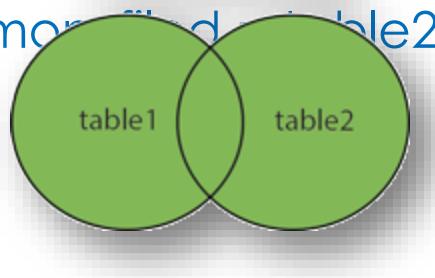
```
SELECT table1.column1, table2.column2...FROM
table1RIGHT JOIN table2ON table1.common_filed
= table2.common_field;
```



# Full Join Syntax

- ▶ The SQL FULL JOIN combines the results of both left and right outer joins.
- ▶ The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.
- ▶ SYNTAX:

```
SELECT table1.column1, table2.column2...FROM table1FULL JOIN
table2ON table1.common_field = table2.common_field;
```



SQL has many built-in functions for performing calculations on data. They are divided into 2 categories:

1. Aggregate Function
2. Scalar Function



# Aggregate Function

These functions are used to do operations from the values of the column and a single value is returned.

AVG() - Returns the average value

COUNT() - Returns the number of rows

FIRST() - Returns the first value

LAST() - Returns the last value

MAX() - Returns the largest value

MIN() - Returns the smallest value

SUM() - Returns the sum



# Student Table

344

## 1. Avg():

Syntax: SELECT AVG(column\_name) FROM table\_name;

Example:

SELECT AVG(AGE) AS AvgAge FROM Students;

Output: AvgAge

19.4

| ID | NAME    | MARKS | AGE |
|----|---------|-------|-----|
| 1  | Harsh   | 90    | 19  |
| 2  | Suresh  | 50    | 20  |
| 3  | Pratik  | 80    | 19  |
| 4  | Dhanraj | 95    | 21  |
| 5  | Ram     | 85    | 18  |

## 2. COUNT():

Syntax: SELECT COUNT(column\_name) FROM table\_name;

Example: SELECT COUNT(\*) AS NumStudents FROM Students;

Output: NumStudents

5



# Student Table

345

## 3. First():

Syntax: SELECT FIRST(column\_name) FROM table\_name;

Example:

SELECT I MarksFirst RKS) AS MarksFirst FROM Students;

Output

90

| ID | NAME    | MARKS | AGE |
|----|---------|-------|-----|
| 1  | Harsh   | 90    | 19  |
| 2  | Suresh  | 50    | 20  |
| 3  | Pratik  | 80    | 19  |
| 4  | Dhanraj | 95    | 21  |
| 5  | Ram     | 85    | 18  |

## 4. LAST():

Syntax: SELECT LAST(column\_name) FROM table\_name;

Example: S MarksLast ST(MARKS) AS MarksLast FROM

Students;

Output: 82



# Student Table<sup>346</sup>

## 5. MAX():

Syntax: SELECT MAX(column\_name) FROM table\_name;

Example:

SELECT MAX(MARKS) AS MaxMarks FROM Students;

Output: MaxMarks

---

95

---

| ID | NAME    | MARKS | AGE |
|----|---------|-------|-----|
| 1  | Harsh   | 90    | 19  |
| 2  | Suresh  | 50    | 20  |
| 3  | Pratik  | 80    | 19  |
| 4  | Dhanraj | 95    | 21  |
| 5  | Ram     | 85    | 18  |

6. MIN(): Similar to Max() we can use MIN() function.

## 7. SUM():

Syntax: SELECT SUM(column\_name) FROM table\_name;

Example: SELECT SUM(MARKS) AS TotalMarks FROM Students;

Output: TotalMarks

---

400

---



# Scalar functions:

- ▶ These functions are based on user input, these too returns single value.
  - ▶ UCASE() - Converts a field to upper case
  - ▶ LCASE() - Converts a field to lower case
  - ▶ MID() - Extract characters from a text field
  - ▶ LEN() - Returns the length of a text field
  - ▶ ROUND() - Rounds a numeric field to the number of decimals specified
  - ▶ NOW() - Returns the current system date and time
  - ▶ FORMAT() - Formats how a field is to be displayed



# Student Table<sup>348</sup>

## 1. UCASE()

Syntax: SELECT UCASE(column\_name) FROM table\_name;

Example:

SELECT UCASE(NAME) FROM Students;

Output:

| NAME    |
|---------|
| HARSH   |
| SURESH  |
| PRATIK  |
| DHANRAJ |
| RAM     |

| ID | NAME    | MARKS | AGE |
|----|---------|-------|-----|
| 1  | Harsh   | 90    | 19  |
| 2  | Suresh  | 50    | 20  |
| 3  | Pratik  | 80    | 19  |
| 4  | Dhanraj | 95    | 21  |
| 5  | Ram     | 85    | 18  |

2. LCASE(): Similar to UCASE() we can use LCASE() function.



# Student Table

### 3. MID()

Syntax: SELECT MID(column\_name,start,length) AS some\_name FROM table\_name;

specifying length is optional here, and start signifies start position ( starting from 1 )

Example:

SELECT MID(NAME,1,4) FROM Students;

Output:

| NAME |
|------|
| HARS |
| SURE |
| PRAT |
| DHAN |
| RAM  |

### 4. LEN():

Syntax: SELECT LENGTH(column\_name) FROM table\_name;

Example: SELECT LENGTH(NAME) FROM Students;

| ID | NAME    | MARKS | AGE |
|----|---------|-------|-----|
| 1  | Harsh   | 90    | 19  |
| 2  | Suresh  | 50    | 20  |
| 3  | Pratik  | 80    | 19  |
| 4  | Dhanraj | 95    | 21  |
| 5  | Ram     | 85    | 18  |

| NAME |
|------|
| 5    |
| 6    |
| 7    |
| 3    |



# Student Table

## 5. ROUND():

Syntax: SELECT ROUND(column\_name,decimals)

FROM table\_name;

decimals- number of decimals to be fetched.

Example:

SELECT ROUND(MARKS,0) FROM table\_name;

Output:

MARKS

90

50

80

95

85

| ID | NAME    | MARKS | AGE |
|----|---------|-------|-----|
| 1  | Harsh   | 90    | 19  |
| 2  | Suresh  | 50    | 20  |
| 3  | Pratik  | 80    | 19  |
| 4  | Dhanraj | 95    | 21  |
| 5  | Ram     | 85    | 18  |

## 6. NOW():

Syntax: SELECT NOW() FROM table\_name;

Example: SELECT NAME, NOW() AS DateTime FROM Students;

| NAME    | DateTime             |
|---------|----------------------|
| HARSH   | 1/13/2017 1:30:11 PM |
| SURESH  | 1/13/2017 1:30:11 PM |
| PRATIK  | 1/13/2017 1:30:11 PM |
| DHANRAJ | 1/13/2017 1:30:11 PM |
| RAM     | 1/13/2017 1:30:11 PM |



# Student Table

351

## 7. FORMAT():

Syntax: SELECT FORMAT(column\_name) FROM table\_name;

Example:

```
SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date
FROM Student
```

Output:

| NAME    | Date       |
|---------|------------|
| HARSH   | 2017-01-13 |
| SURESH  | 2017-01-13 |
| PRATIK  | 2017-01-13 |
| DHANRAJ | 2017-01-13 |
| RAM     | 2017-01-13 |

| ID | NAME    | MARKS | AGE |
|----|---------|-------|-----|
| 1  | Harsh   | 90    | 19  |
| 2  | Suresh  | 50    | 20  |
| 3  | Pratik  | 80    | 19  |
| 4  | Dhanraj | 95    | 21  |
| 5  | Ram     | 85    | 18  |



- ▶ A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
  - ▶ So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
  - ▶ You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
- 
- ▶ **To create procedure, use syntax:**

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

- ▶ **To execute created procedure, use syntax:**
- ```
EXEC procedure_name;
```

Refer This Example

- ▶ **Cursor Example:**
 - ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Cursor/example>
- ▶ **Stored Procedure with one parameter:**
 - ▶ <https://github.com/TopsCode/Software-Engineering/tree/master/SQL/Cursor>
- ▶ **Stored Procedure with multiple parameter:**
 - ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Cursor/multipleParam>

Trigger

- ▶ A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs
- ▶ For example, a trigger can be invoked when a row is inserted into a specified table.

Syntax:

```
create trigger [trigger_name]
```

```
[before | after]
```

```
{insert | update | delete}
```

```
on [table_name]
```

```
[for each row]
```

```
[trigger_body]
```



Explanation of syntax:

create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.

[before | after]: This specifies when the trigger will be executed.

{insert | update | delete}: This specifies the DML operation.

on [table_name]: This specifies the name of the table associated with the trigger.

[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

[trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is

Refer This Example

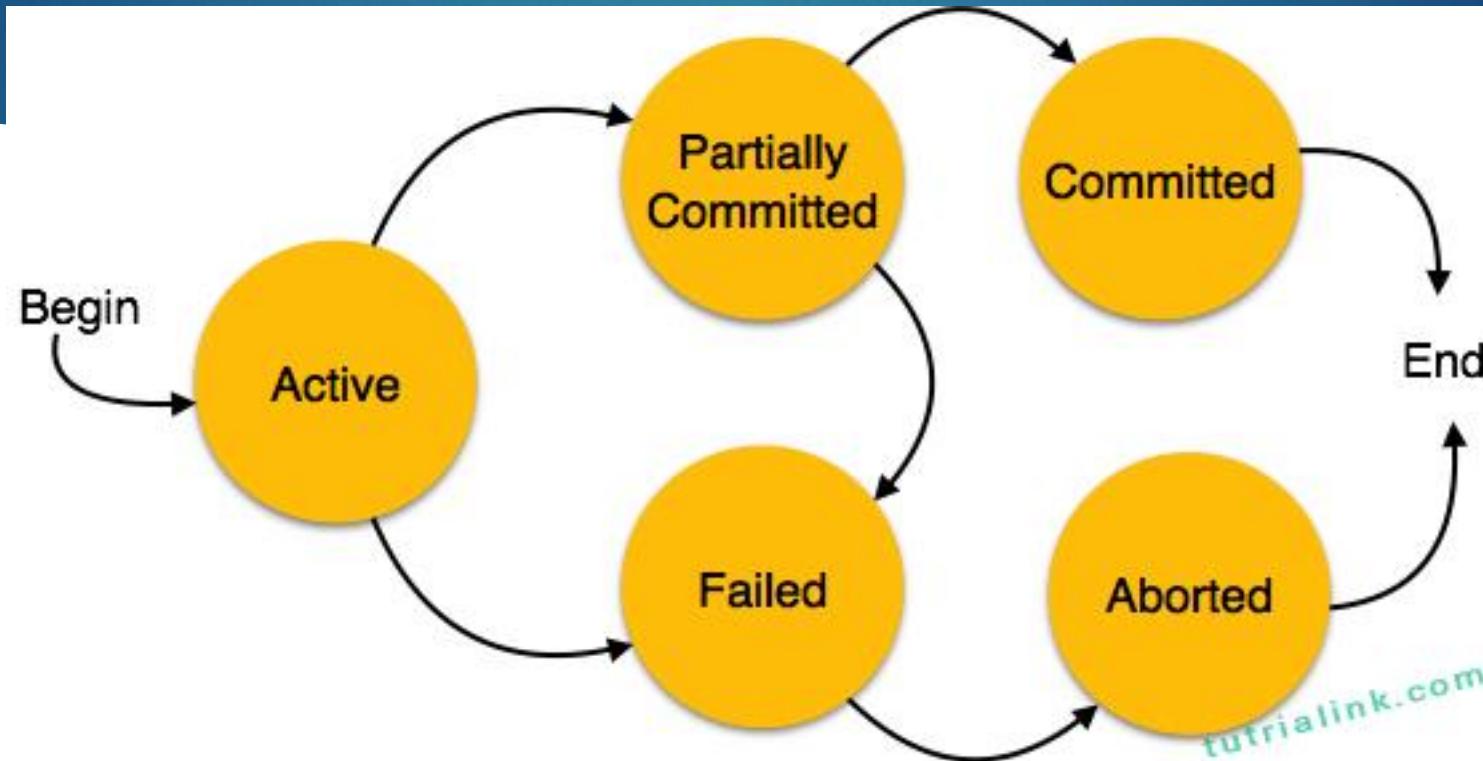
- ▶ **Before Trigger:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Trigger/beforeTrigger>

- ▶ **After Trigger:**
- ▶ <https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Trigger/afterTrigger>



Transaction

- A transaction is a logical unit of work of database processing that includes one or more database access operations.
- A transaction can be defined as an action or series of actions that is carried out by a single user or application program to perform operations for accessing the contents of the database.
- The operations can include retrieval, (Read), insertion (Write), deletion and modification.
- Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure.
- In order to maintain consistency in a database, before and after transaction, certain properties are followed. These are called **ACID** properties(**A**tomicity, **C**onsistency, **I**solati



1. Atomicity:

- ▶ This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none.
- ▶ There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- ▶ For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

2. Consistency:

- The database must remain in a consistent state after any transaction.
- No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well..
- For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

3. Isolation

- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.
- For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

4. Durability

- The database should be durable enough to hold all its latest updates even if the system fails or restarts.
- If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
- If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action..
- For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed



The following commands are used to control transactions.

- ▶ **COMMIT** – to save the changes.
- ▶ **ROLLBACK** – to roll back the changes.
- ▶ **SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK.

1. **Commit:**

- ▶ The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.
- ▶ The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.
- ▶ The syntax for the COMMIT command is as follows:



2. Rollback:

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows –

```
ROLLBACK;
```

3. Savepoint:

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINT SAVEPOINT_NAME;
```

This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below

```
ROLLBACK TO SAVEPOINT_NAME;
```



Cursor

- ▶ It is a temporary area for work in memory system while the execution of a statement is done.
- ▶ A Cursor in SQL is an arrangement of rows together with a pointer that recognizes a present row.

- ▶ It is a database object to recover information from a result set one row at once.

- ▶ It is helpful when we need to control the record of a table in a singleton technique, at the end of the day one row at any given moment. The arrangement of columns the cursor holds as the dynamic set.

Main components of Cursors

Each cursor contains the followings 5 parts,

Declare Cursor: In this part we declare variables and return a set of values.

`DECLARE cursor_name CURSOR FOR SELECT_statement;`

Open: This is the entering part of the cursor.

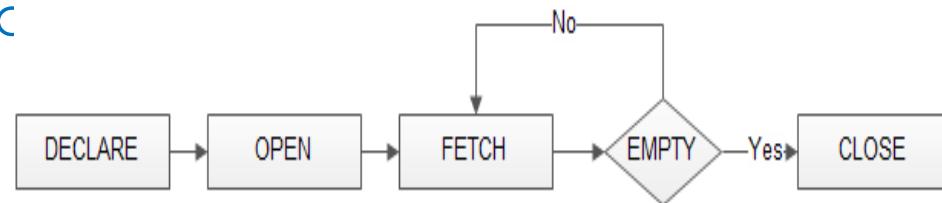
`OPEN cursor_name;`

Fetch: Used to retrieve the data row by row from a cursor.

`FETCH cursor_name INTO variables list;`

Close: This is an exit part of the cursor and used to close a cursor.

`CLOSE c`



Syntax:

```
DECLARE variables;  
records;  
create a cursor;  
BEGIN  
OPEN cursor;  
FETCH cursor;  
process the records;  
CLOSE cursor;  
END;
```



- Database Backup is storage of data that means the copy of the data.
- It is a safeguard against unexpected data loss and application errors.
- It protects the database against data loss.
- If the original data is lost, then using the backup it can be reconstructed.
- The backups are divided into two types,
 1. Physical Backup
 2. Logical Backup

1. Physical backups

- Physical Backups are the backups of the physical files used in storing and recovering your database, such as datafiles, control files and archived redo logs, log files.
- It is a copy of files storing database information to some other location, such as disk, some offline storage like magnetic tape.
- Physical backups are the foundation of the recovery mechanism in the database.
- Physical backup provides the minute details about the transaction and modification to the database.

2. Logical backup:

- Logical Backup contains logical data which is extracted from a database.
- It includes backup of logical data like views, procedures, functions, tables, etc.
- It is a useful supplement to physical backups in many circumstances but not a sufficient protection against without physical backups, because logical backup provides only logical consistency.



Importance of Backups

- Planning and testing backup helps against failure of media, operating system, software and any other kind of failures that cause a serious data crash.
- It determines the speed and success of the recovery.
- Physical backup extracts data from physical storage (usually from disk to tape). Operating system is an example of physical backup.
- Logical backup extracts data using SQL from the database and store it in a binary file.
- Logical backup is used to restore the database objects into the database. So the logical backup utilities allow DBA (Database Administrator) to back up and recover selected objects in the database.



Causes of Failure

1. System Crash

- ▶ System crash occurs when there is a hardware or software failure or external factors like a power failure.
- ▶ The data in the secondary memory is not affected when system crashes because the database has lots of integrity. Checkpoint prevents the loss of data from secondary memory.

2. Transaction Failure

- ▶ The transaction failure is affected on only few tables or processes because of logical errors in the code.
- ▶ This failure occurs when there are system errors like d~~a~~ or unavailability of system resources to execute the transaction.



3. Network Failure

A network failure occurs when a client – server configuration or distributed database system are connected by communication networks.

4. Disk Failure

- Disk Failure occurs when there are issues with hard disks like formation of bad sectors, disk head crash, unavailability of disk etc.

5. Media Failure

- Media failure is the most dangerous failure because, it takes more time to recover than any other kind of failures.
- A disk controller or disk head crash is a typical example of media failure.



Recovery

- ▶ Recovery is the process of restoring a database to the correct state in the event of a failure.
- ▶ It ensures that the database is reliable and remains in consistent state in case of a failure.

Database recovery can be classified into two parts;

1. **Rolling Forward** applies redo records to the corresponding data blocks.
2. **Rolling Back** applies rollback segments to the datafiles. It is stored in transaction tables.

