-> Pointer: is a variable which holds the address of another variable

ex: int *p;

i. Start with the variable.
ii. Move ahead upto the Delimiter [semicolon]
iii. Get Back step-by-step

eg:1. int *p; -> p is a pointer to an integer.
2. int **p; -> p is a pointer to pointer to integer.
3. int ***p; -> p is a pointer to pointer to pointer to integer
4. int *p[5] -> p is an array of 5 pointers to integer.
5. int **p[5] -> p is an array of 5 pointers to pointer to an integer.
6. int (*p)[5] -> p is a pointer to an Array of 5 integers

* Use of pointer: pointer provides the programmer an alternative approach to access memory.

1. Data pointer: holds the address of data.
2. Function pointer: holds the address of function.

* Create a data/function pointer.
* Assign the address of data/function respectively.
* Dereference the data/function pointer.

-> * operator in C :

1. '*' can multiply — a = b * c ;
2. '*' can create a pointer — int *p; void (*p)(int,int);
3. '*' can dereference a pointer — d = *p;

-> Alternative Approach to access data :

a | 50

1. create a data pointer	int *p;
2. Assign address of data	p = &a;
3. Dereference data pointer	printf("%d", *p);

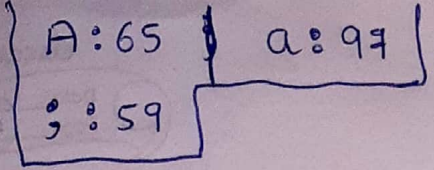
→ Dereferencing operator.

-> Alternative Approach to access function :

1. create a function pointer	void (*p)(int, int);
2. Assign address of function	p = exmp1; or p = &exmp1;
3. Dereference function pointer	(*p)(15,20);

=> &a provides the address of data 'a'.
=> exmp1 provides the address of function body exmp1()

```
void exmp1 (int a, int b)
{
 - - - -
  - - -
}
```



→ pointer Size: irrespective of the data type of a pointer two bytes is allocated on most compilers.

Q1 How to print a statement without using a semicolon?

Ans: Include the statement as a condition within the if-control construct.

Q2 How to print a semicolon without using semicolon?

Ans: Use the ASCII code of semicolon i.e., 59 as
printf("%c", 59);

Q3 How to execute Both if block & else block simultaneously?

Ans: → components of the if-else control construct:
1. condition: expression evaluating to true or false
2. if block: contains statements
3. else block: contains statements.

```
if (!print("Hello"))
{
}
else
{
    printf("world");
}
```

* The return type of printf() is an integer.

1. Zero → treated as false
2. Non-zero → True.

30, 40, -50 } are treated as TRUE

0 } treated as FALSE.

Q4. How to execute multiple statements in if & else blocks simultaneously ?

Ans:

```
#include<stdio.h>
void main()
{
  if(!printf("Hello"), printf("world"), printf("!"))
  {

  }
  else
  {
     printf("\n Nice to Meet ");
     printf(" You.");
  }
}
```

Output:

```
Hello world!
Nice to Meet You.
```

-> STEPS:

1. Remove all the statements from if-block & place them in the if-condition
2. Replace semicolon with comma
3. Apply the ! symbol.

Qs. How to execute both if block & else block simultaneously in Java?

Ans: STEPS:

1. Remove the statement in if block & place it in if-condition.
2. Remove the semicolon in the if-condition.
3. Apply == null in the if condition.

```
class Demo
{
  public static void main(String args[])
  {
    if(System.out.println("Hello world") == null)
    {

    }
    else
    {
    System.out.println("welcome, to java");
    }
  }
}
```

Output:

```
Hello world
welcome, to java.
```

-> pre-increment : First increment than use.
-> post-increment : First use than increment.

Ex:
```
int a=5;
int b=++a;
printf("%d,%d", a,b);   // 6,6
```

Ex
```
int a=7;
int b=a++;
printf("%d,%d", a,b);   // 8,7
```

-> Object file vs Executable file :

Ex: add.c -> after compilation => add.obj
↓ source file
↓ object file
↓ Machine Specific code [windows compatible]

* C & C++ compilers are platform dependent or OS dependent.
* object file + library file : Executable file
↓ coded in Machine-level language
* Linker takes the object file & the required library as input & produces an executable file [add.exe]
* Loader takes the data from the hard-disk & would be putting it into the RAM.

-> C & C++ fault lines:

* C & C++ are platform dependent and hence not suitable to code internet or web based applications.

-> Addition & subtraction without using (+ & - operat-or respectively):

1. Addition: int a=6, b=7;
int c=a+b; //13
⇓
c=a-(-b); //13

2. Subtraction: int a=12, b=6;
int c=12-6; //6
⇓
c=a+~b+1; //6

-> Java data types:

byte -> 1 Byte
short -> 2 Bytes
int -> 4 Bytes
long -> 8 Bytes

-> Negative numbers are stored in 2's complement Binary representation

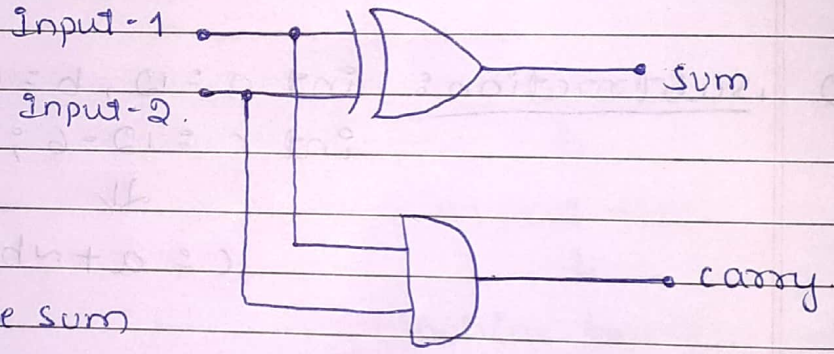
-> Quotient & Remainder:

* RULE:

1. sign of the Quotient -> sign of numerator * sign of denominator.

2. sign of the remainder -> sign of the numerator.

-> How to perform Arithmetic operation without using any arithemetic operator : in JAVA.



* XOR gate gives the sum
* AND gates gives the carry.

-> XOR gate : output is 1 for different i/p & 0 for same i/p.

Input 1	Input2	carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

-> Below is the program to perform addition of two integers without using any arithmetic operator

```
class Demo
{
  public static void main (String args[])
  {
     int input1 = 42;
     int input2 = 23;
     int sum = 0;
     int carry = 0;

     while (input2 != 0)
     {
        sum = input1 ^ input2;
        carry = input1 & input2; //~input & input2;
        input1 = input1 ^ input2;   for Subtraction.
        input2 = carry << 1;
     }

    System.out.printf("%d", sum);
  }
}
```

→ Swapping two integers without using 3rd variable:

* Logic: (1): a = a+b;
  b = a-b;
  a = a-b;
  } Addition & Subtraction operators approach

(2): a = a*b;
b = a/b;
a = a/b
} Multiplication & Division operators approach.

* Limitations of the above approaches are :

1. May Fail due to OVERFLOW.
2. May Fail due to Exception [if one of the input is equal to zero (0)]

-> Swapping of Two numbers using Bitwise Operators in Java :

* Logic :
```
a = a^b;  // EX-OR the input thrice
b = a^b;  // swapps two number.
a = a^b;  // These is the most Efficient
          // Approach.
```

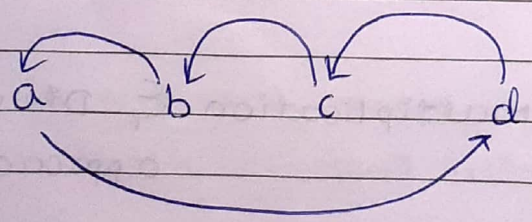
-> Swapping in a Single Statement in JAVA :

1. Logic : (1) : `a = a+b-(b=a);`
   (2) : `a = a*b/(b=a);`
   (3) : `a = a^b^(b=a);`

   } single statement is more efficient than the 3 statement approach

-> Swapping three variables, four variables in Java :

* Logic :
```
a = a^b^(b=a);
b = b^c^(c=b);
c = c^d^(d=c);
```



⇒ How to Swap Nibbles in a Byte:

Steps: 1. & number with Hexadecimal OF. [Extracts Right Nibble]
2. & number with Hexadecimal FO. [Extracts the left Nibble]
3. << by 4 positions
4. >> by 4 positions
5. combine nibbles using | [OR] operator.

* Logic: a=(num & 0x0F)<<4 | (num & 0xF0)>>4;

⇒ Swapping odd & Even Bits in a Byte in JAVA;

Steps: 1: & number with Hexadecimal 55. [Extracts the odd Bits]
2: & number with Hexadecimal AA. [Extract the Even Bits]
3: << by 1 position.
4: >> by 1 position.
5: combine odd & even bits using | operator.

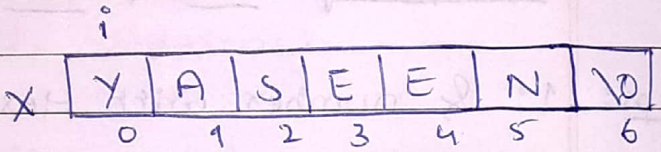
* Logic: a = (num&55)<<1 | (num && AA)>>1;

STRINGS:

* In C strings are treated as Array of characters terminated by NULL (\0) character.
* In Java strings are treated as objects.

computing length of a string : [in C]:

* As long as (X[i] is not equal to '\0')
{
  increment count
  increment i
}
print count



count = 6

```
#include <stdio.h>
void main()
{
  char x[] = "Md Yaseen";
  int count = 0;
  int i = 0;

  while (x[i] != '\0')
  {
    ++count;
    ++i;
  }
  printf("%d", count); // 9
}
```

→ Using Built-in library function the Above program can be written as:

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

void main()
{
  char x[]= " Md Yaseen ";
  printf("%d", strlen(x));     //9
}
```

→ computing length of strings : [in JAVA]:

* There are 2 challenges:

1. No direct Access to data
2. No '\0' at the end.

```
class Demo
{
  public static void main(String args[])
  {
    String x = "Md Yaseen";
    x = x.concat("\0");
    char y[] = x.toCharArray();
    int count =0, i=0;

    while(y[i] != '\0')
    {
      ++count;
      ++ i;
```

```
    }
    System.out.printf("%d", count);
  }
}
```

=> Using Built-in method:

```
class Demo
{
  public static void main (String args[])
  {
    String x = "Md yaseen";
    System.out.println("%d", x.length());
  }
}
```

=> copying one String into another in JAVA:

Steps: 1. create a String object with 'x' as the reference.
2. Extract the data from String object & place it in a character Array 'y'.
3. Compute the length of Array 'y'.
4. Create a new Array 'a' with its size same as 'y'.
5. Create an index variable 'i' & initialize it to 0.
6. Copy the data from 'y' Array to 'a' Array.

```
As long as (i != size)
{
  place the y[i] in a[i]
  increment i
}
```

-> JAVA code for the Above steps: [To copy a string]

```
class copystr
{
  public static void main(String args[])
  {
    String x = "Md Yaseen";
    char y[] = x.toCharArray();
    int size = y.length;
    char a[] = new char[size];
    int i=0;

    while(i != size)
    {
      a[i] = y[i];
      ++i;
    }
    System.out.println(y);   // Md Yaseen
    System.out.println(a);   // Md Yaseen
  }
}
```

-> Reverse a String in JAVA:

```
while(i != size)
{
  a[size-1-i] = y[i];
  ++i;
}
```

*. Remaining is same as the above program for copying the string.

-> foreach loop in JAVA:

ex:
```
int sum = 0;
for (int data : a)
{
    sum = sum + data;
}
System.out.printf("%d", sum);
```

-> Map Data structure:

* Data in a map is stored as a {key, value} pair.

Definition: MAP is a data structure which holds the data as the <key, value> pair.

=> Properties:

1. Keys are Unique
2. values need not to be Unique

EX: DATA = YASEEN

MAP:

Key	value
Y	1
A	1
S	1
E	2
N	1

* Types of Map Supported By JAVA:

1. HashMap
2. TreeMap
3. LinkedHashMap