# Fundamentals of Programming
# CCS1063/CSE1062
# Lecture 2

Professor Noel Fernando

# Contents

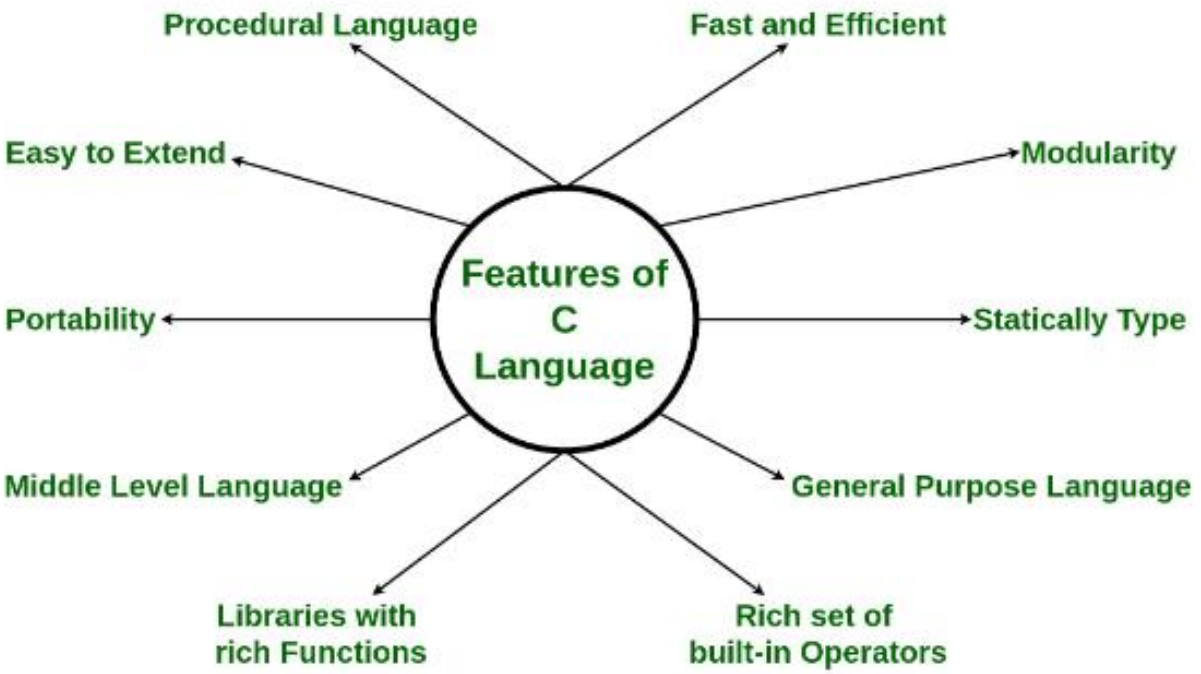
- History of C Language
- Flow charts and pseudo codes
- Problem solving using Flow charts and pseudocodes
- Bit Patterns
- Binary Numbers
- Data Type formats
- Character representation
- Integer  Representation
- Floating point number representation

# History of C Language

- **C programming language** was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), U.S.A.
- It was developed to overcome the problems of previous languages such as B, BCPL, etc.
- Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL.

# Features of C Language

- Simple
- Machine Independent or Portable
- Mid-level programming language (it supports the feature of both low-level and high-level languages)
- structured programming language
- Rich Library
- Memory Management
- Fast Speed
- Pointers
- Recursion
- Extensible



# Programming fundamentals

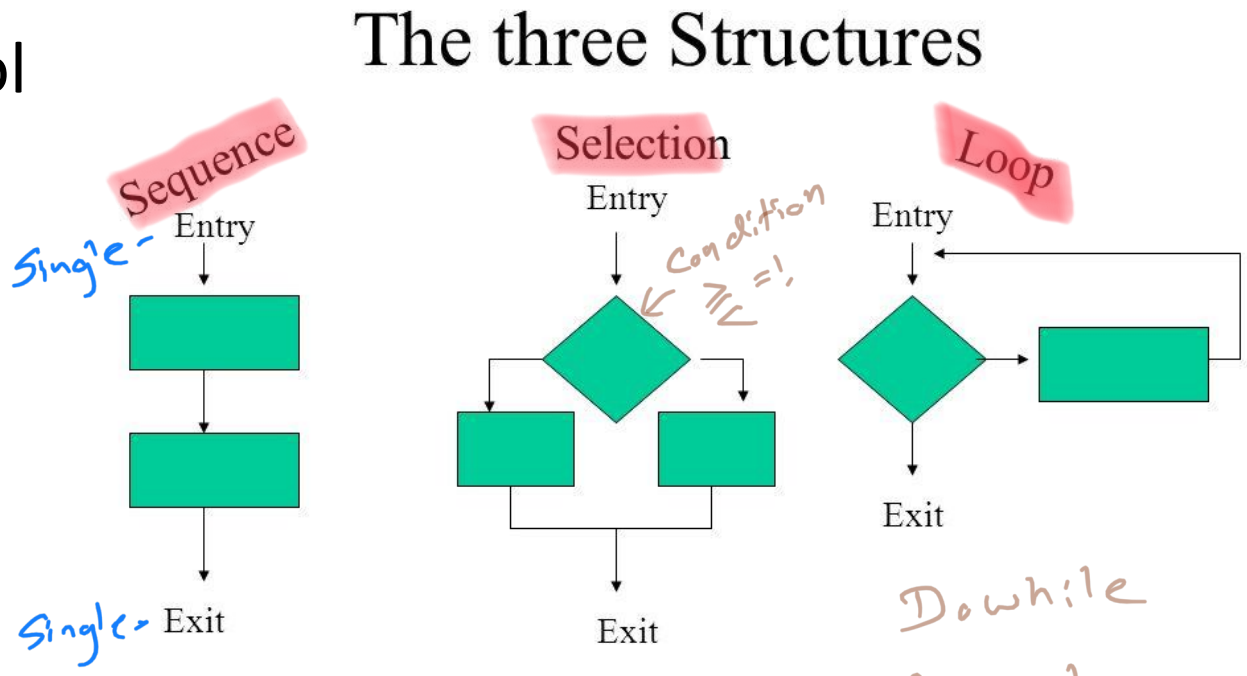
- Algorithm
- It is a formula, a recipe or a step by-step procedure to be followed in order to obtain the solution to a problem.
- To be useful as a basis for writing program.
- The algorithm must;
  - Arrive at a correct solution within a finite time.
  - Be clear, precise and unambiguous.
  - Be in a format which lends itself to an elegant implementation in a programming language.

# Control Structures

- The key to elegant algorithm design lies in limiting the control structures to only three constructs.

1. Sequence
2. Iteration
3. Selection

Always only one entry.




one after other statment in order.

Dowhile

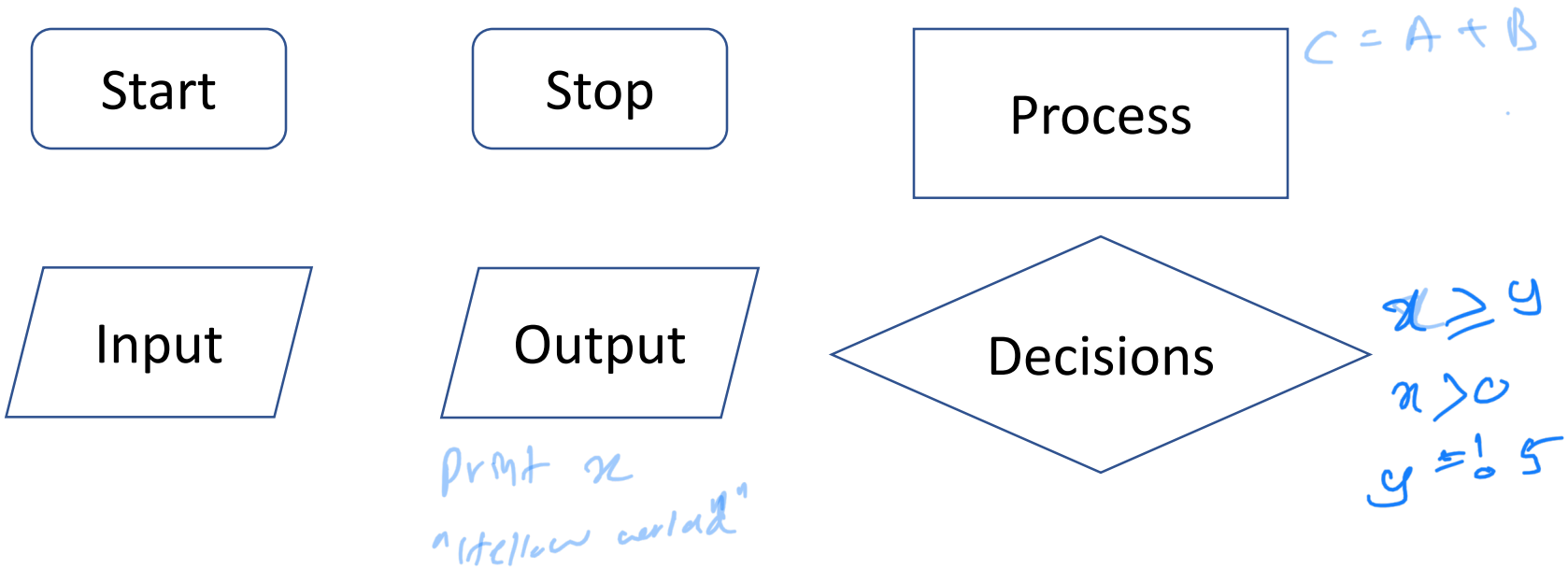
Repeat

for

# Flow Charts

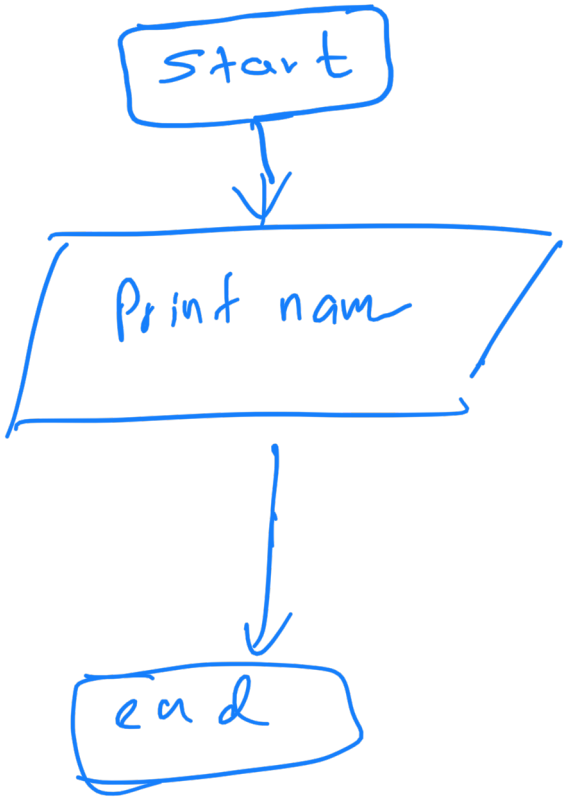
- Flow charts can be used as a way of expressing algorithms.

- Note that there are standard symbols to indicate:



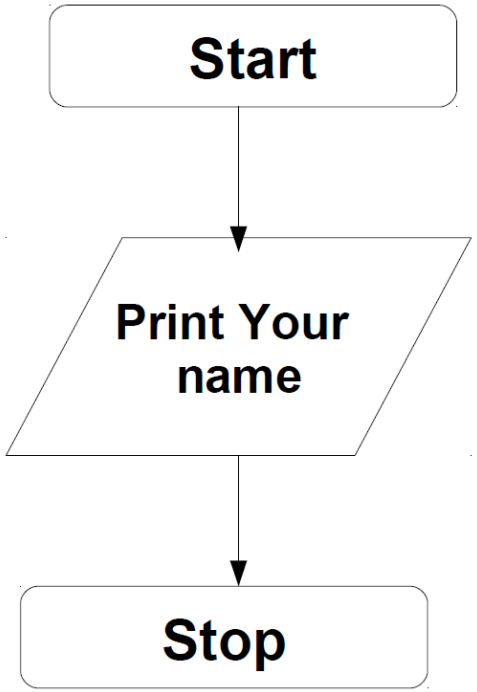
# Problem 1:

- Draw a flow chat to print your name.



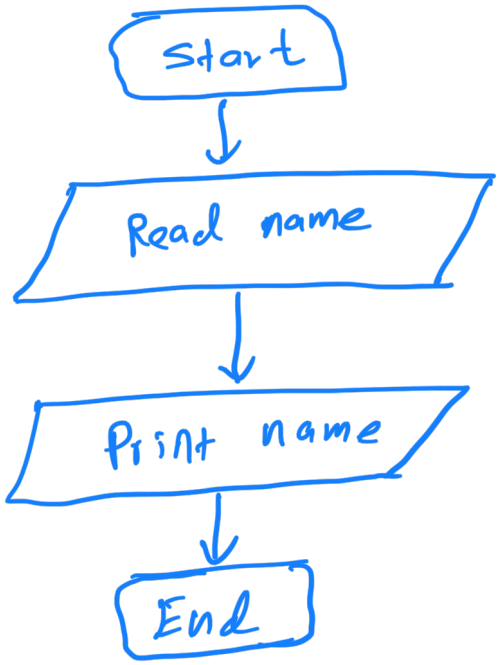
# Problem 1:

Draw a flow chat to print your name.



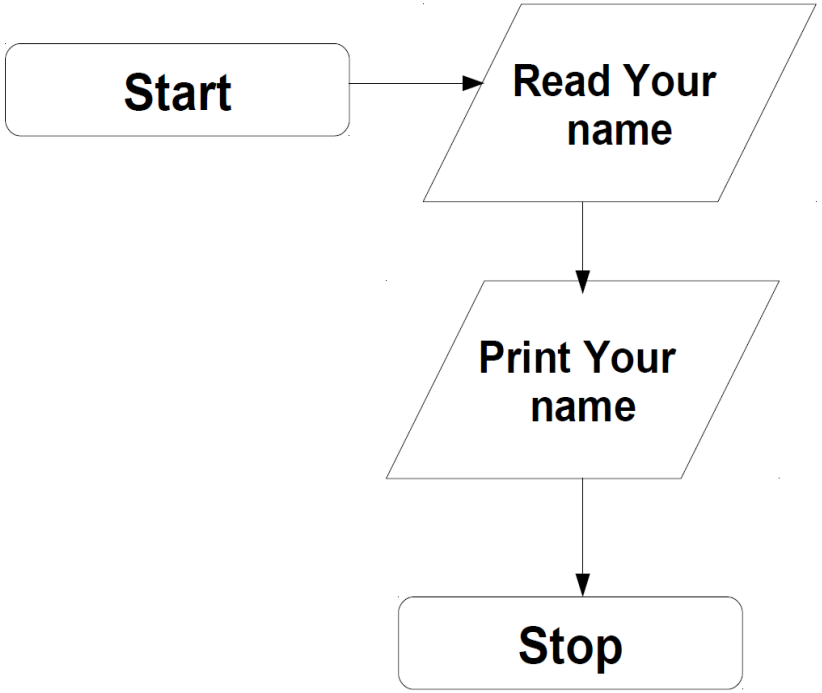
# Problem 2:

- Draw a  flaw chart  to read and print your name.



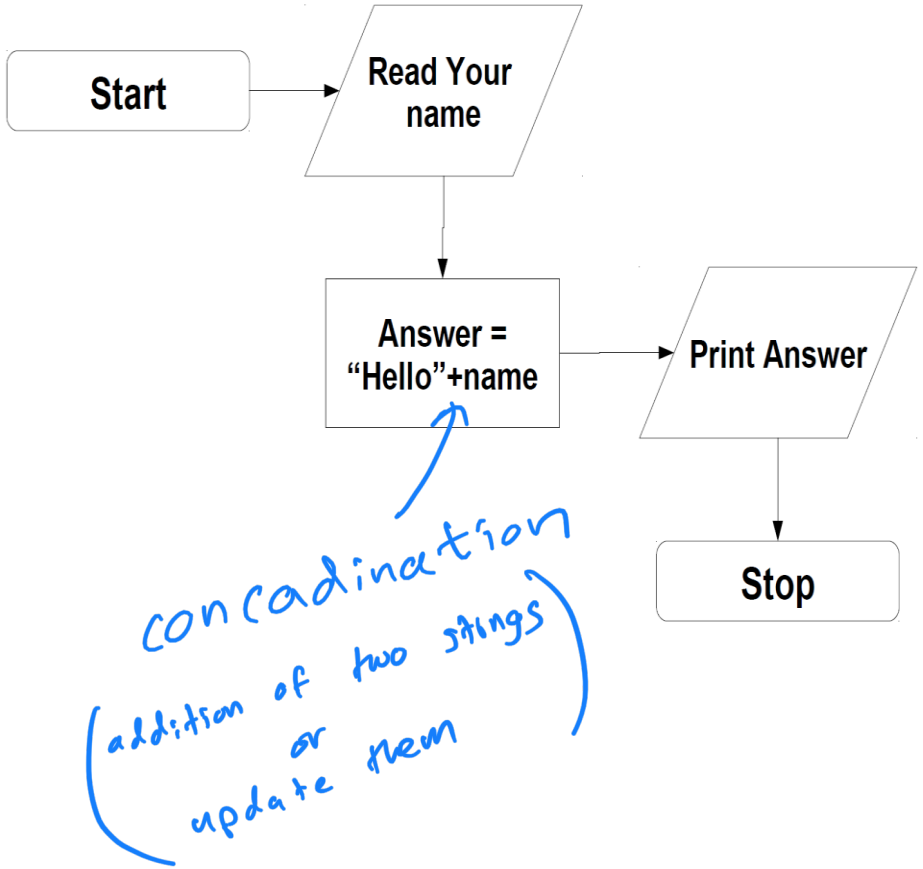
# Problem 2:

- Draw a  flaw chart  to read and print your name.



# Problem 3:

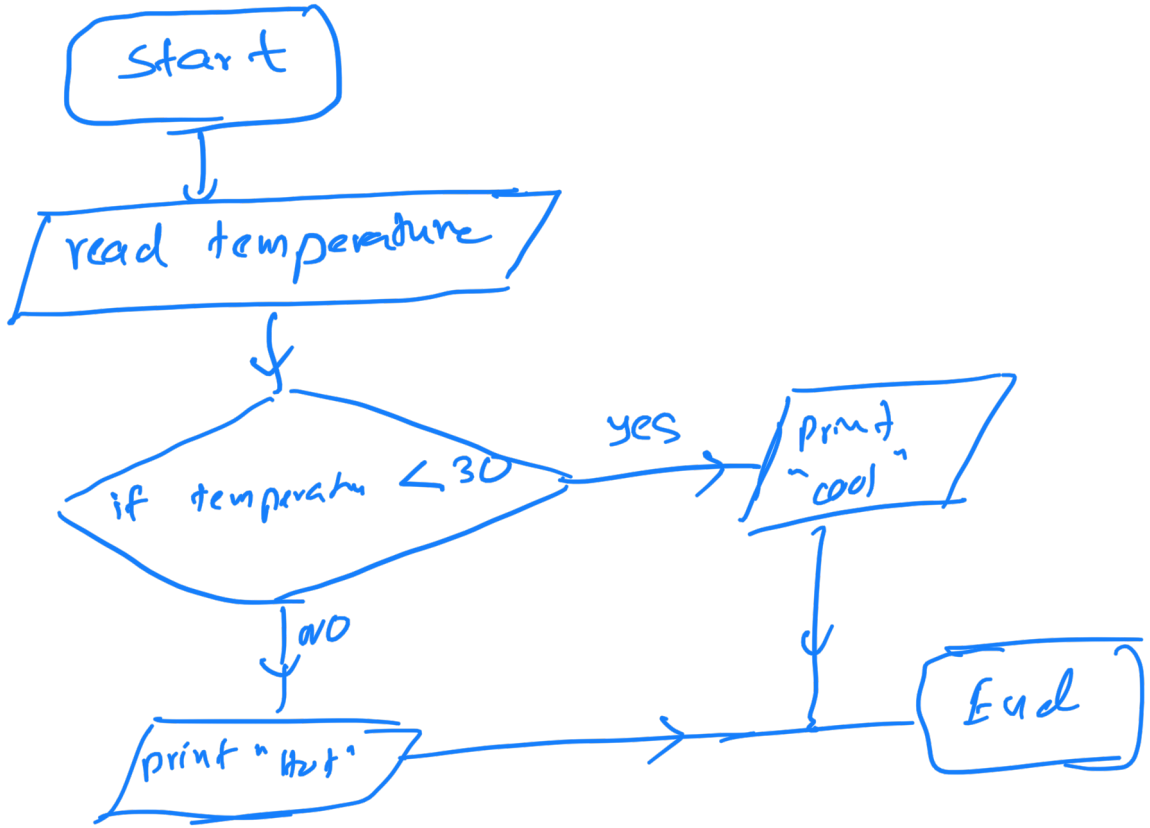
Write a program to read and say “Hello” to your name



# Problem 4:

Write a program to read temperature and print “Cool” or “Hot”

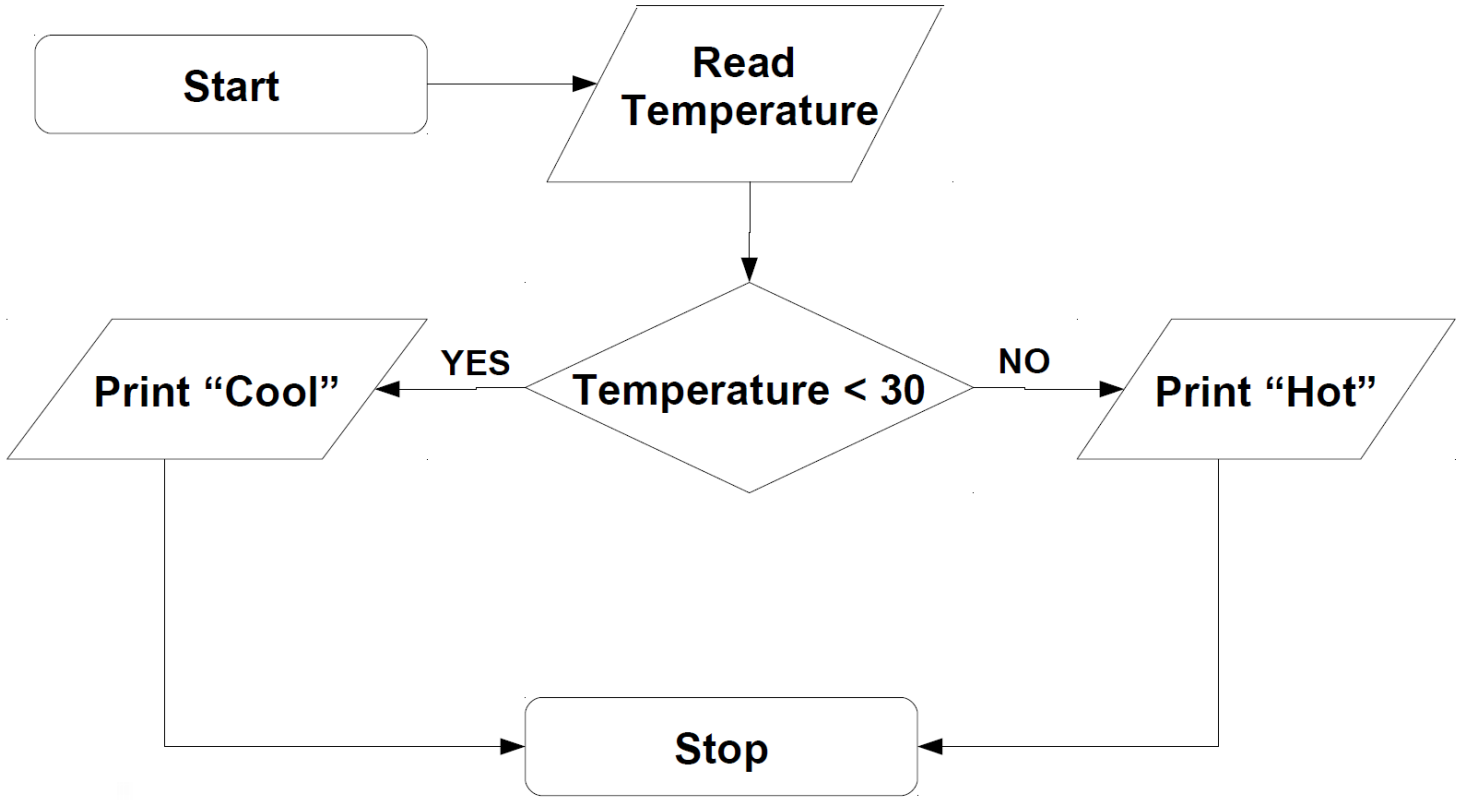
- If temperature is less than 30: Cool Otherwise: Hot



# Problem 4:

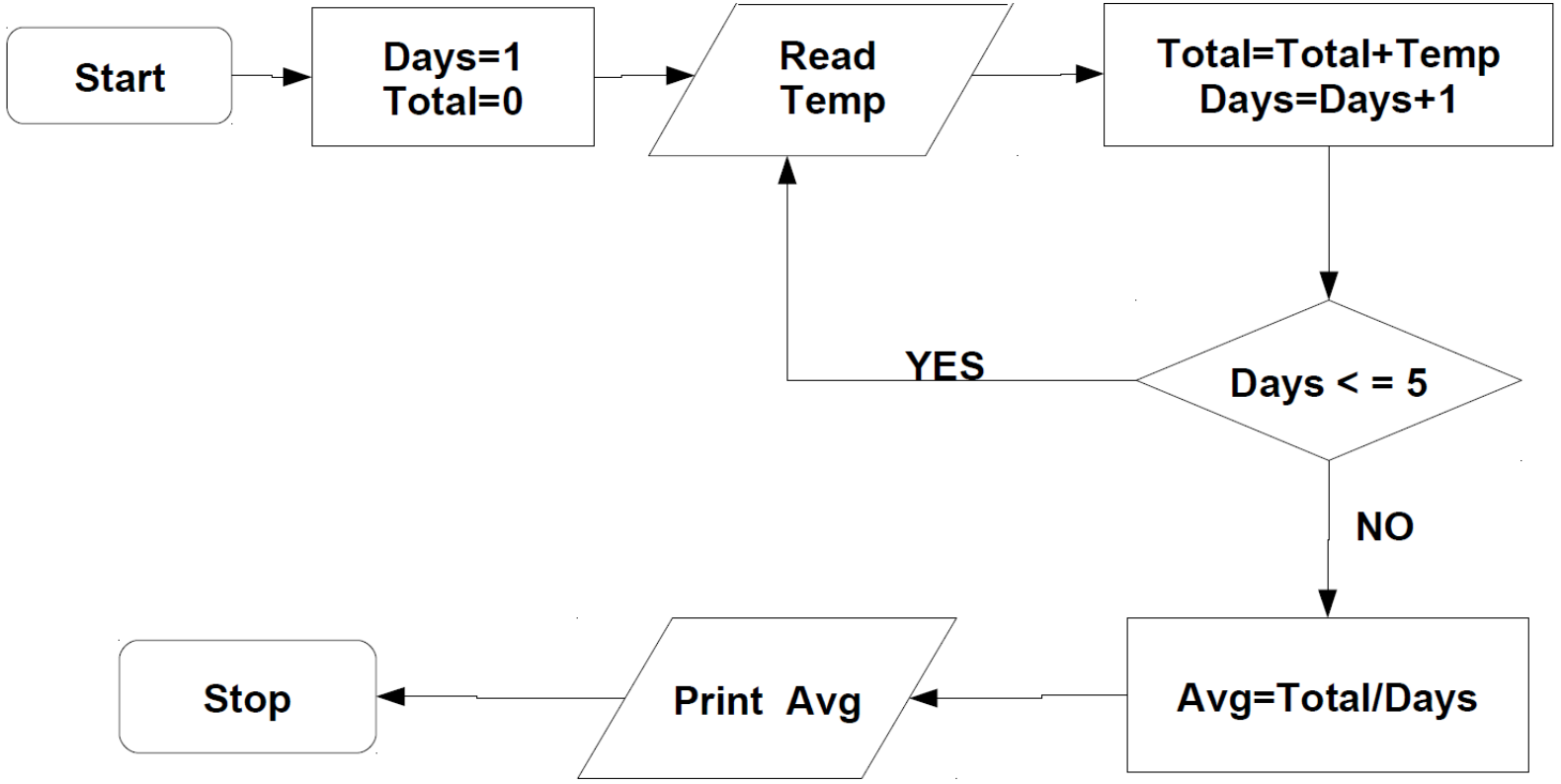
Write a program to read temperature and print “Cool” or “Hot”

- If temperature is less than 30: Cool Otherwise: Hot



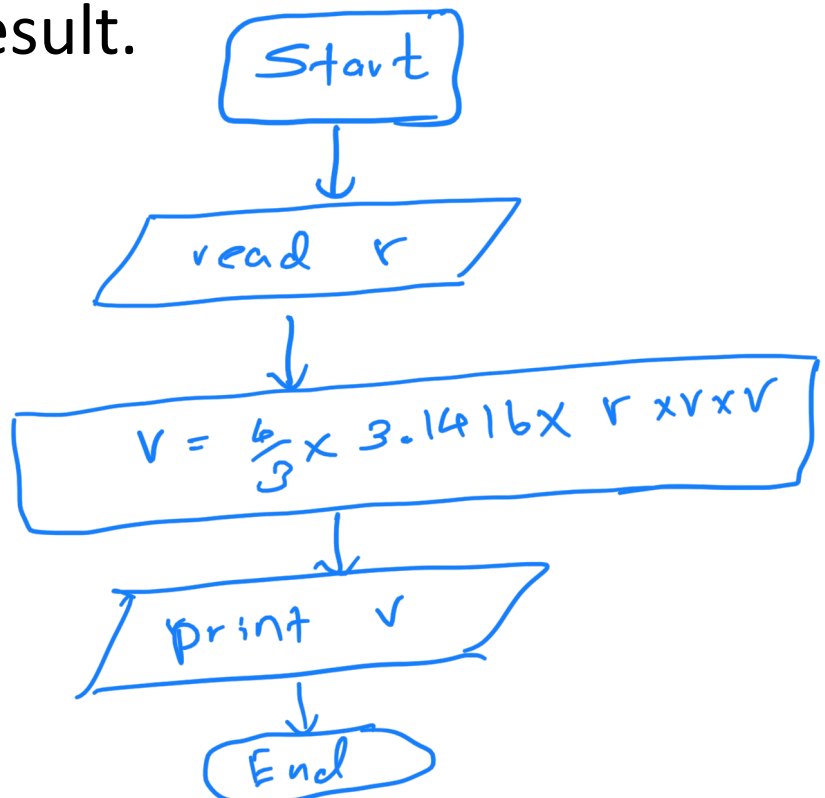
# Problem 5:

Read temperature for 5 days and print average.



# Problem 6:

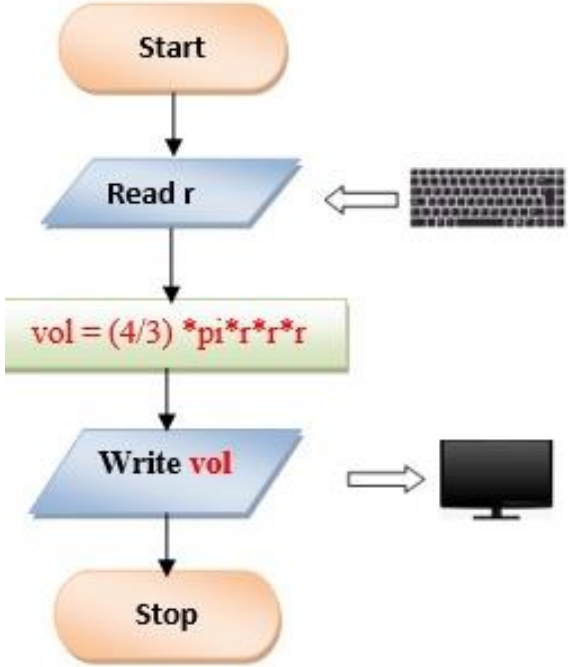
- Draw a flaw chart and write a pseudocode algorithm to compute the volume of a sphere. Use the formula: V = (4/3) *pi*$r^3$ where pi is equal to 3.1416 approximately. The **r** is the radius of sphere. Display the result.



# Problem 6:

- Draw a flaw chart and write a pseudocode algorithm to compute the volume of a sphere. Use the formula: V = (4/3) *pi*$r^3$ where pi is equal to 3.1416 approximately. The **r** is the radius of sphere. Display the result.

**Flowchart**

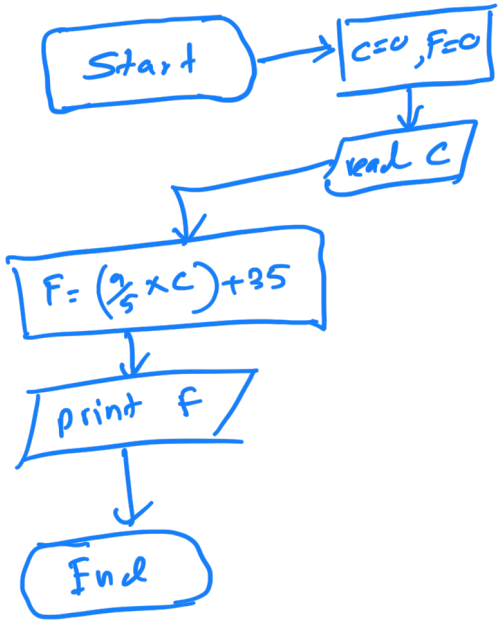



**Algorithm**

1. Start
2. Read r
3. vol = (4/3) *pi*r*r*r
4. Print or display vol
5. Stop

# Problem 7:

- Draw a flow chart and write a pseudo code algorithm to converts the input Celsius degree into its equivalent Fahrenheit degree. Use the formula: F = (9/5) *C+32.



# Problem 7:

- Draw a flow chart and write a pseudo code algorithm to converts the input Celsius degree into its equivalent Fahrenheit degree. Use the formula: F = (9/5) *C+32.

**Flowchart**

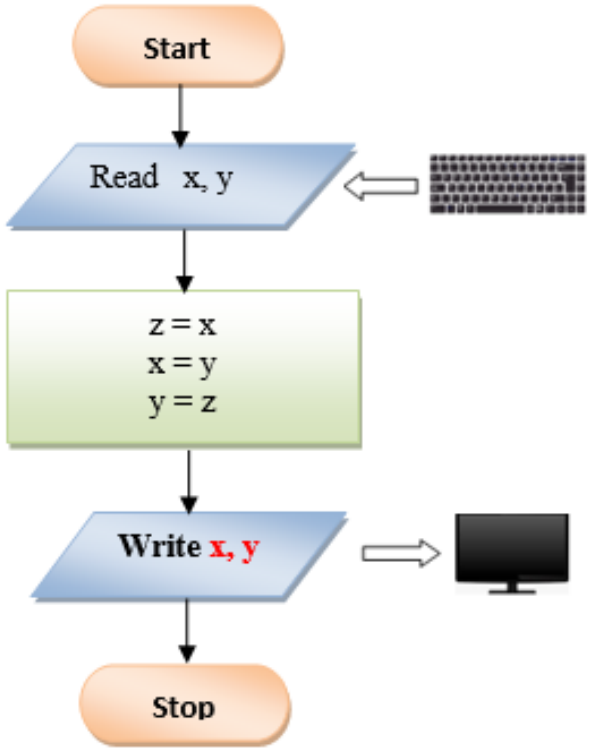

**Algorithm**

1. Start
2. Initialize F=0, C=0
3. Read C
4. Fh = (1.8*C) + 32
5. Print or display Fh
6. Stop

# Problem 8:

- Draw a flow chart and Write a pseudocode algorithm to exchanges the value of two variables: x and y. The output must be: the value of variable y will become the value of variable x, and vice versa.

**Flowchart**




**Algorithm**

1. Start
2. Read x, y
3. Declare third variable, z

   **z = x**

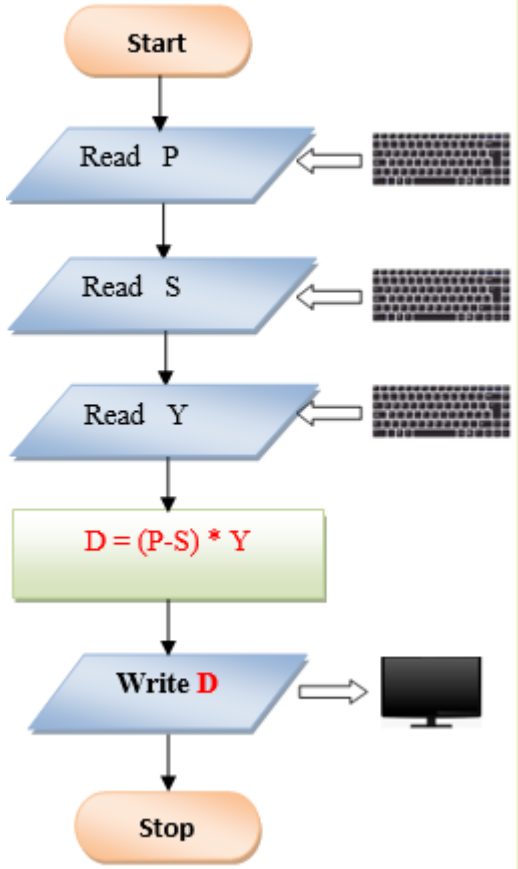
   **x = y**

   **y = z**
4. Print or display x, y
5. Stop

# Problem 9:

- Write a program that takes as input the purchase price of an item (P), its expected number of years of service (Y) and its expected salvage value (S). Then outputs the yearly depreciation for the item (D). Use the formula: D = (P – S) Y.

**Flowchart**




**Algorithm**

1. Start
2. Read P
3. Read S
4. Read Y
5. D = (P-S) * Y
6. Print or display D
7. Stop

# Problem 10

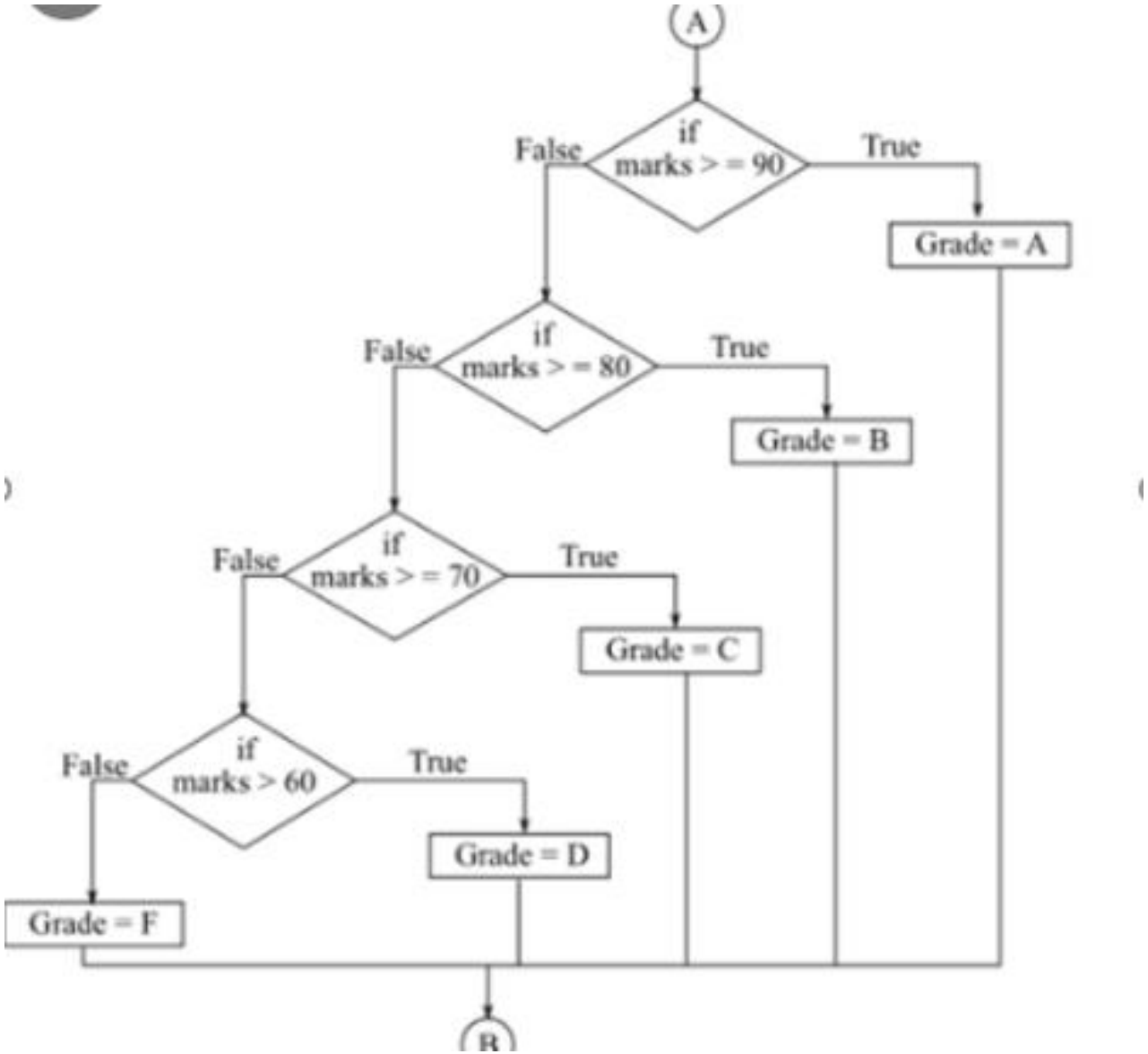
- Draw a flow chart, Write pseudo code algorithm and finally calculate the Grade
- Hint : marks >=90  “A GRADE”

  marks >=80 and =< 89 “B GRADE”

  marks >=7 0and =< 79 “C GRADE”

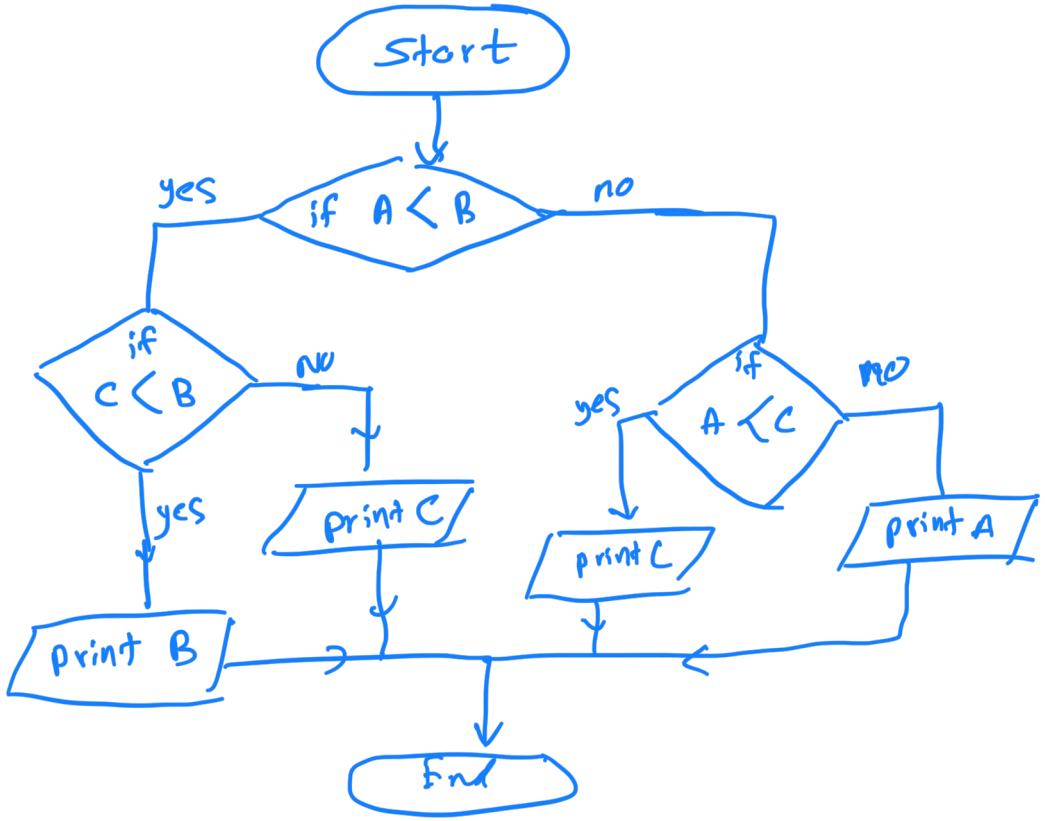
  marks >=60 and =< 69 “D” GRADE”

  Otherwise “ F GRADE”


A
if
marks > = 90
False
True
Grade = A
if
marks > = 80
False
True
Grade = B
if
marks > = 70
False
True
Grade = C
if
marks > 60
False
True
Grade = D
Grade = F
B

# Problem 10:

Write Pseudocode to print the height of the tallest boy in a three-guy basketball team. Their height is stored as A, B, C.



# Problem 10:

Write Pseudocode to print the height of the tallest boy in a three-guy basketball team. Their height is stored as A, B, C.

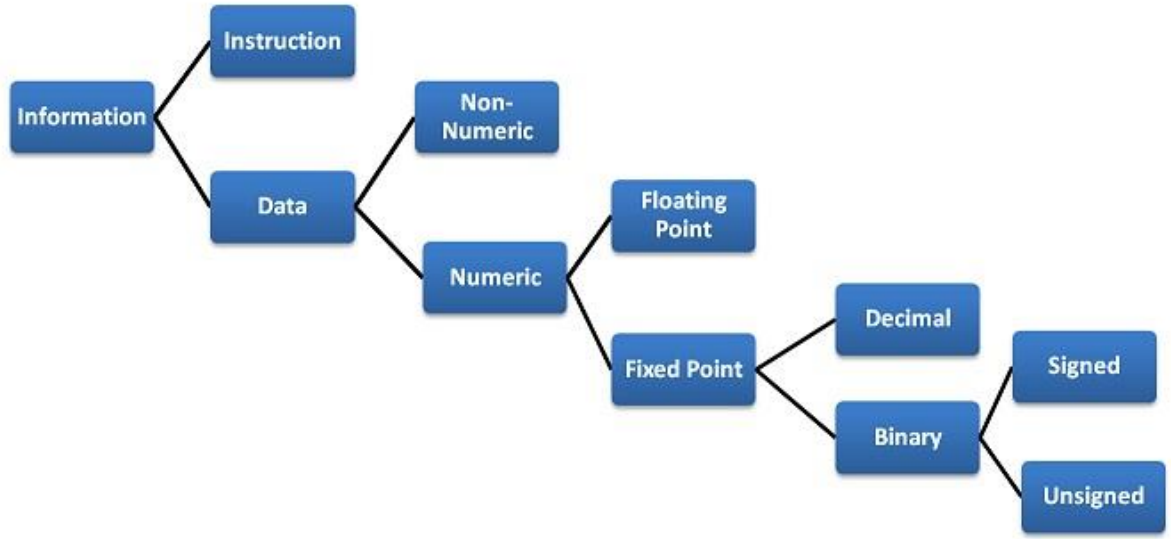
- SOLUTION In the following algorithm, it's assumed that the first boy A is the tallest, then the second boy B is compared to this tallest. If the second happens to be taller than the assumed tallest, the second is stored as the new tallest, otherwise the first number is still the tallest. Algorithm does the same with the third boy.

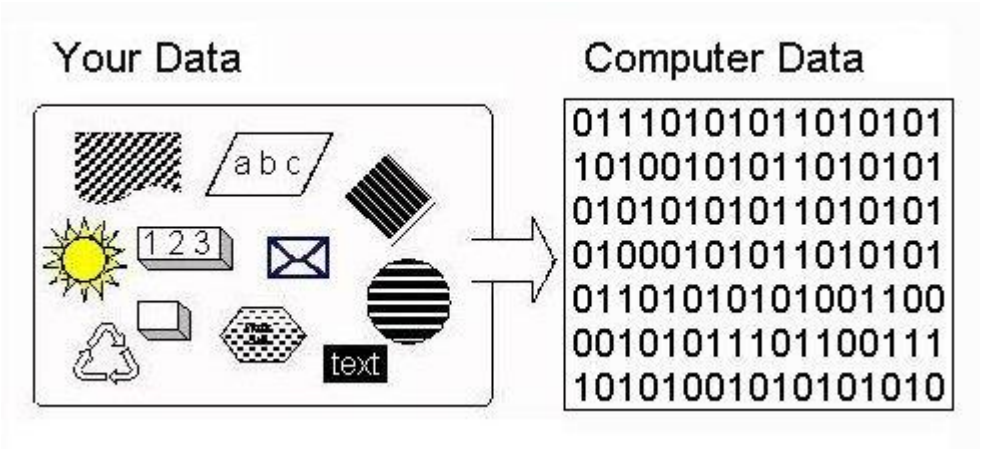
```
SOLUTION
READ A, B, C
SET Tallest = A

IF B > Tallest
        SET Tallest = B
ENDIF
IF C > Tallest
        SET Tallest = C
ENDIF
PRINT 'The Tallest is ', Tallest
```

# Data representation

- Data representation refers to the manner in which data is stored in the computer.
- There are several different formats for the data storage.
- It is important for computer problem solvers to understand the basic formats



# Why is  it Important

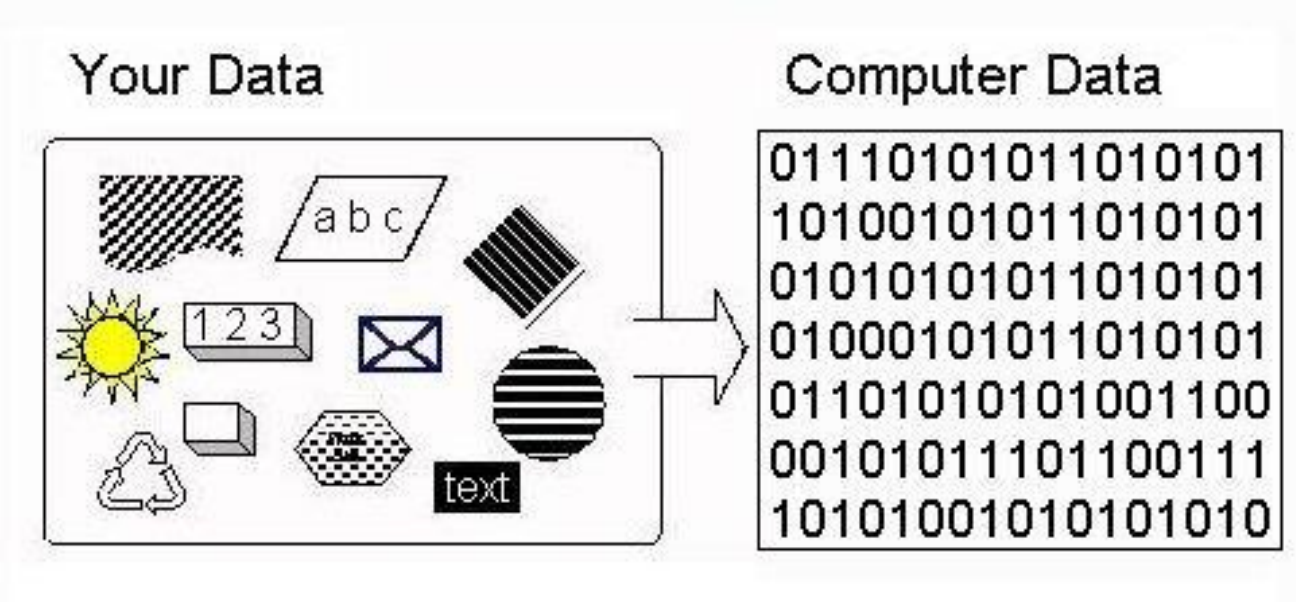
- As an example:
- We will learn that since we have finite storage, it is possible to overflow a storage location by trying to store to large a number.
- Most programming languages provide multiple data types each providing different length storage for variables
- It is up to the programmer  to choose the data type with a length that won't overflow.
- Knowing how numbers are represented in storage helps one to understand this.

## Entire Data types in c:

| Data type | Size(bytes) | Range | Format string |
|---|---|---|---|
| Char | 1 | -128 to 127 | %c |
| Unsigned char | 1 | 0 to 255 | %c |
| Short or int | 2 | -32,768 to 32,767 | %i or %d |
| Unsigned int | 2 | 0 to 65535 | %u |
| Long | 4 | -2147483648 to 2147483647 | %ld |
| Unsigned long | 4 | 0 to 4294967295 | %lu |
| Float | 4 | 3.4 e-38 to 3.4 e+38 | %f or %g |
| Double | 8 | 1.7 e-308 to 1.7 e+308 | %lf |
| Long Double | 10 | 3.4 e-4932 to 1.1 e+4932 | %lf |

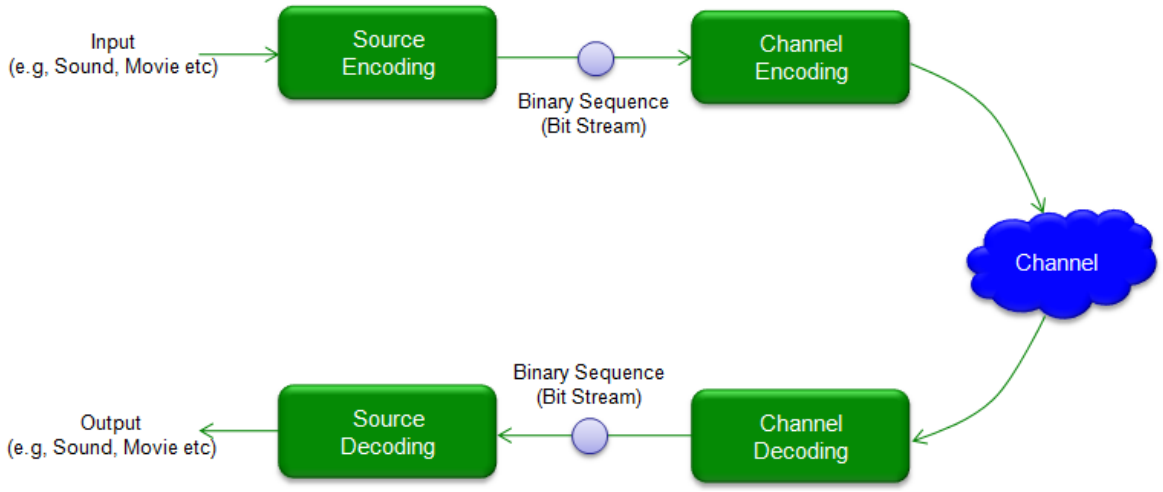
# Bit pattern

- As you recall from an earlier slides , data may like various forms: characters, numbers, graphical, etc.
- All data is stored in the computer as a sequence of Bits (binary digits)
- This is a Universal storage format for all data types, and it is called a bit pattern.



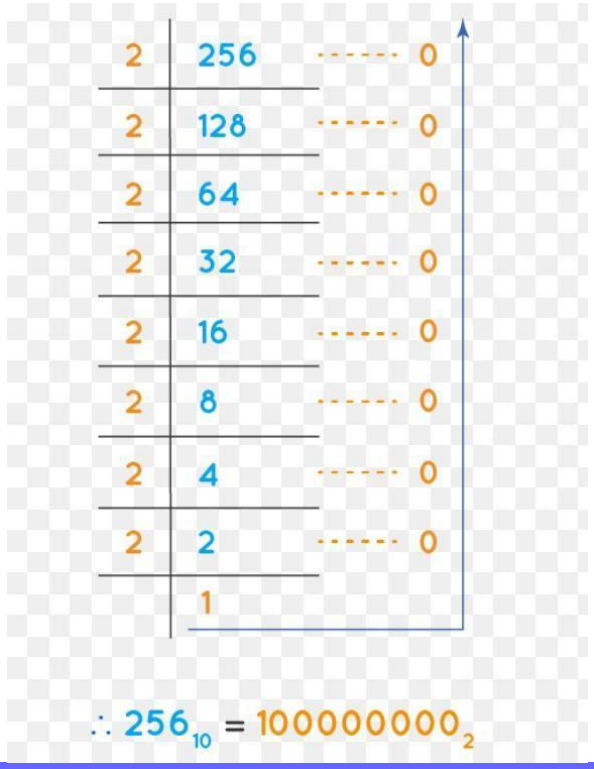
# Bits and Bytes

- A bit  is the smallest unit of data stored in a computer and it has a value of 0 or1.
- It is like a switch on(1) or Off(0)
- In computers, bis are stored electronically in RAM and auxiliary storage derive by two state digital circuits.
- The storage device itself doesn’t know what the bit pattern represents, but software (application S/W, O/S, and I/O device firmware) stores and interprets the pattern
- That data is coded then stored and when retrieved it is decoded.
- A byte is a string of a 8 bis and is called a character when data is text.



# Binary Numbers

- Each Bit pattern is a binary number, that is a number represented by 0's and 1's rather than 0,1,2,3…..9 as decimal numbers are
- For example, bit pattern like 1010 and 11011 are also binary numbers
- Binary numbers are based on powers of 2 rather than powers of 10 as decimal numbers are




- For example, if one tries to store 256 in 1-byte there is overflow because the largest value storable in 8 bits is 255 as one can see from the following table:

| Binary number: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| Powers of 2: | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| Decimal value: | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

- Note that 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255

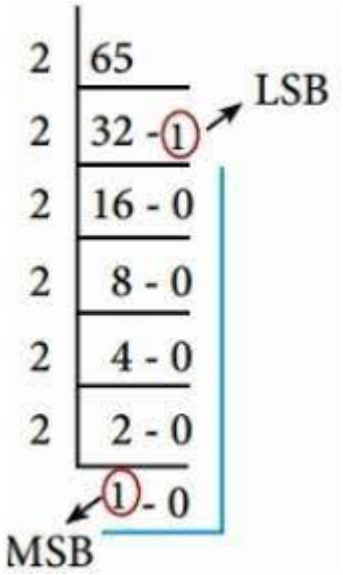
# Data Types format

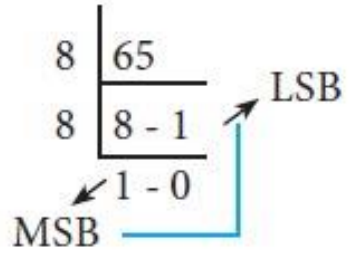
- As we have learned, fundamentally all data  is stored as a bit pattern
- But the different data types have different bit pattern formats

- We want to learn the formats for:
  - Characters (for example, left, Lane, a, ?, \)
  - Integers (for example, 1, 453, -10, 0)
  - Floating point numbers (for example, 3.14159, 1.2, -567.235, 0.009)

# Character representation

- The  American standard code for information interchange (ASCII) is the scheme used to assign a bit pattern to each of the characters
- ASCII charts come in different flavors:
- Some have 7 bits string, some 8 or more
- Some show the binary code for the various characters as well as the code represented in other number systems e.g. decimal, hexadecimal and octal
- For example, the letter A has the ASCII code:
- 1000001 in n Binary, 65 in decimal, 41 in hexadecimal and 101 in Octal




$(65)_{10} = (1\ 0\ 1)_8$

```
Decimal Number 65 It Is  Hexadecimal: 41
```

# Subset of ASCII Chart

| Char | Dec | Bin | Char | Dec | Bin | Char | Dec | Bin | Char | Dec | Bin | Char | Dec | Bin | Char | Dec | Bin |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Space | 32 | 010 0000 | 0 | 48 | 011 0000 | @ | 64 | 100 0000 | P | 80 | 101 0000 | ` | 96 | 110 0000 | p | 112 | 111 0000 |
| ! | 33 | 010 0001 | 1 | 49 | 011 0001 | A | 65 | 100 0001 | Q | 81 | 101 0001 | a | 97 | 110 0001 | q | 113 | 111 0001 |
| " | 34 | 010 0010 | 2 | 50 | 011 0010 | B | 66 | 100 0010 | R | 82 | 101 0010 | b | 98 | 110 0010 | r | 114 | 111 0010 |
| # | 35 | 010 0011 | 3 | 51 | 011 0011 | C | 67 | 100 0011 | S | 83 | 101 0011 | c | 99 | 110 0011 | s | 115 | 111 0011 |
| $ | 36 | 010 0100 | 4 | 52 | 011 0100 | D | 68 | 100 0100 | T | 84 | 101 0100 | d | 100 | 110 0100 | t | 116 | 111 0100 |
| % | 37 | 010 0101 | 5 | 53 | 011 0101 | E | 69 | 100 0101 | U | 85 | 101 0101 | e | 101 | 110 0101 | u | 117 | 111 0101 |
| & | 38 | 010 0110 | 6 | 54 | 011 0110 | F | 70 | 100 0110 | V | 86 | 101 0110 | f | 102 | 110 0110 | v | 118 | 111 0110 |
| ' | 39 | 010 0111 | 7 | 55 | 011 0111 | G | 71 | 100 0111 | W | 87 | 101 0111 | g | 103 | 110 0111 | w | 119 | 111 0111 |
| ( | 40 | 010 1000 | 8 | 56 | 011 1000 | H | 72 | 100 1000 | X | 88 | 101 1000 | h | 104 | 110 1000 | x | 120 | 111 1000 |
| ) | 41 | 010 1001 | 9 | 57 | 011 1001 | I | 73 | 100 1001 | Y | 89 | 101 1001 | i | 105 | 110 1001 | y | 121 | 111 1001 |
| * | 42 | 010 1010 | : | 58 | 011 1010 | J | 74 | 100 1010 | Z | 90 | 101 1010 | j | 106 | 110 1010 | z | 122 | 111 1010 |
| + | 43 | 010 1011 | ; | 59 | 011 1011 | K | 75 | 100 1011 | [ | 91 | 101 1011 | k | 107 | 110 1011 | { | 123 | 111 1011 |
| , | 44 | 010 1100 | < | 60 | 011 1100 | L | 76 | 100 1100 | \ | 92 | 101 1100 | l | 108 | 110 1100 | \| | 124 | 111 1100 |
| - | 45 | 010 1101 | = | 61 | 011 1101 | M | 77 | 100 1101 | ] | 93 | 101 1101 | m | 109 | 110 1101 | } | 125 | 111 1101 |
| . | 46 | 010 1110 | > | 62 | 011 1110 | N | 78 | 100 1110 | ^ | 94 | 101 1110 | n | 110 | 110 1110 | ~ | 126 | 111 1110 |
| / | 47 | 010 1111 | ? | 63 | 011 1111 | O | 79 | 100 1111 | _ | 95 | 101 1111 | o | 111 | 110 1111 | | | |

# Numeric Representation

- ASCII codes are an inefficient method for  representing numbers
- For example, the number 1024 using 8 bit ASCII would require four bytes or 32 bits of storage.
- Arithmetic operations on numbers represented in AII are very complicated
- Representing the precision of a number, that is, the number of digits store, may require large amount of space when stored in ASCII
- There are more efficient schemes for numbers.

# Numeric Representation

- Integer Representation
- Sign and modulus
- One’s complement
- Two’s complement
- Representation of Fractions
- Floating point or real.
- IEEE

# Integer representation

- An integer is a whole number, that is a number without a decimal portion.
- Integers may positive, negative or zero
- A plus-sign ( not required) or minus sign in front of the number is used to represent positive and negative numbers and zero
- There are two categories of integer representation : unsigned and signed

# How can we represent a binary number?

- Binary number can only be represented by a 0 or 1, so we cannot use a (-) to denote a negative number.
- Therefore, we cannot use a specialized symbol so it must be either a 0 or 1.
- There are THREE methods used, they are:
- These are:
  1. Sign-Magnitude method,
  2. 1’s Complement method, and
  3. 2’s complement method.

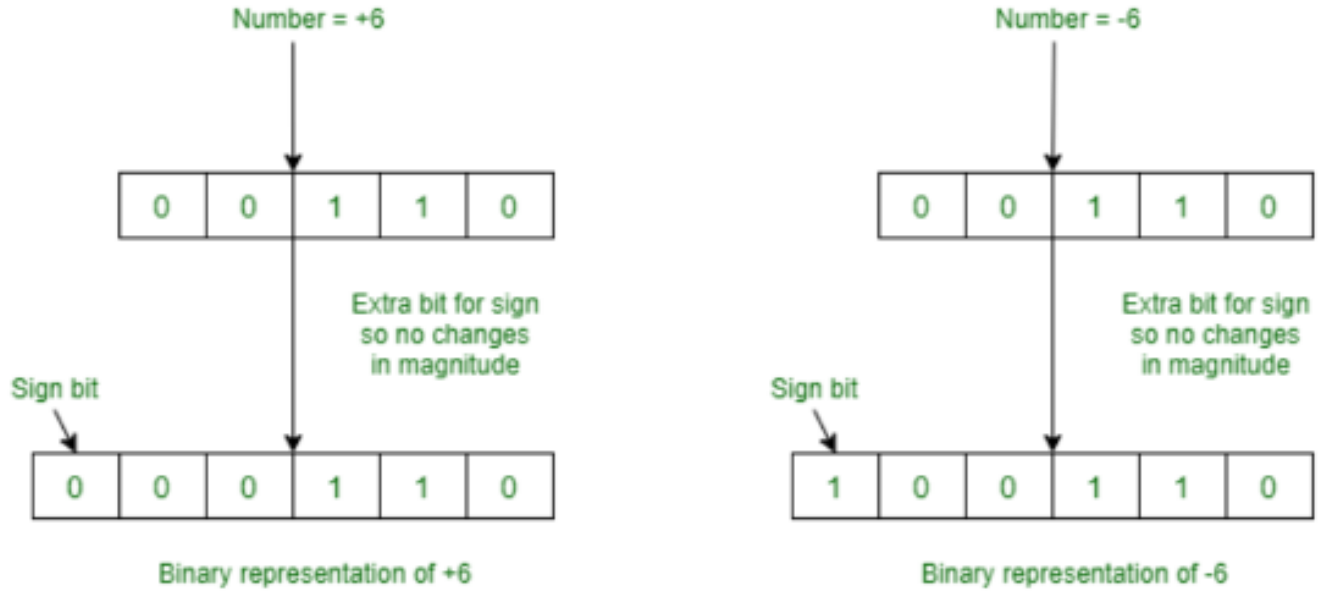
# 1. Signed Magnitude Method

- Add an extra sign bit to recognize negative and positive numbers.
- Sign bit has 1 for negative number and 0 for positive

- E.g. with a 8 bit value

$-28 = 0\ 0011100$

SIGN Modulus




6 bit value

# 1 Signed Magnitude (SM)

- Represent the decimal number as binary
- Left bit (MSB) used as the sign bit
- Only have 7 bits to express the number
- $12_{10} = 00001100$
- $-12_{10} = 10001100$
- How many representations are there for zero?

- For n bits register, MSB will be sign bit and (n-1) bits will be magnitude.
- Then, Negative lowest number that can be stored is $-(2^{(k-1)}-1)$and positive largest number that can be stored is $(2^{(k-1)}-1)$ .
- If n=8,
- MSB for sign digit, remaking 7 digits for magnitude

Range is $-2^{8-1}-1$ to $2^{8-1}-1$

$=-2^{7}-1$ to $2^{7}-1$

$=-(128-1)$ to $(128-1)$

$=-127$ to $127$

An 8-bit sign-magnitude representation, then, can represent any integer from $-127_{(10)}$ to $127_{(10)}$.

# Example 1

- Add the numbers (+5) and (+3) using a computer.
- The numbers are assumed to be represented using 4-bit SM notation.

```
  111  <- carry generated during addition
  0101 <- (+5) First Number
+ 0011 <- (+3) Second Number
  1000 <- (+8) Sum
```

# Example 2

- Add the numbers (-4) and (+2) using a computer.
- The numbers are assumed to be represented using 4-bit SM notation.

```
   000 <- carry generated during addition

   1100 <- (-4) First number

+  0010 <-(+2) Second Number

   1110 <- (-2) Sum
```

Here, the computer has given the wrong answer of -6 = 1110, instead of giving the correct answer of -2 = 1010.

# Disadvantages

- There are two notations for 0(0000 and 1000), which is very inconvenient when the computer wants to test for a 0 result.
- It is not convenient for the computer to perform arithmetic.
- Due to the above mention ambiguities, SM notation is generally not used to represent signed numbers inside a computer.

# 2. One's Complement

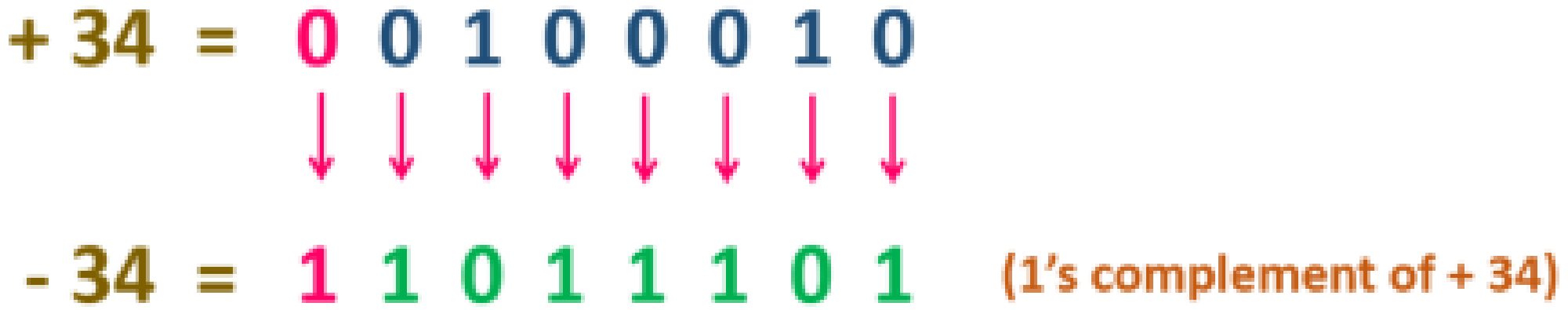
- In digital electronics, the binary system is one of the most common number representation techniques.
- As its name suggest binary number system deals with only two number 0 and 1, this can be used by any device which is operating in two states.
- one's complement is toggling or exchanging all the 0's into 1 and all the 1's into 0 of any number.

- **Method: Invert the ones and zeros**

- **$11_{10}$ = 00001011**
- **$-11_{10}$ = 11110100**

- **0 in MSB implies positive**
- **1 in MSB implies negative**

# E.g 1. Write the one's complement of -34 in 8-bit 1's complement form



# E.g. 2 - Represent -60 in 8-bit 1’s complement form

**To represent -60 in 1’s complement form**

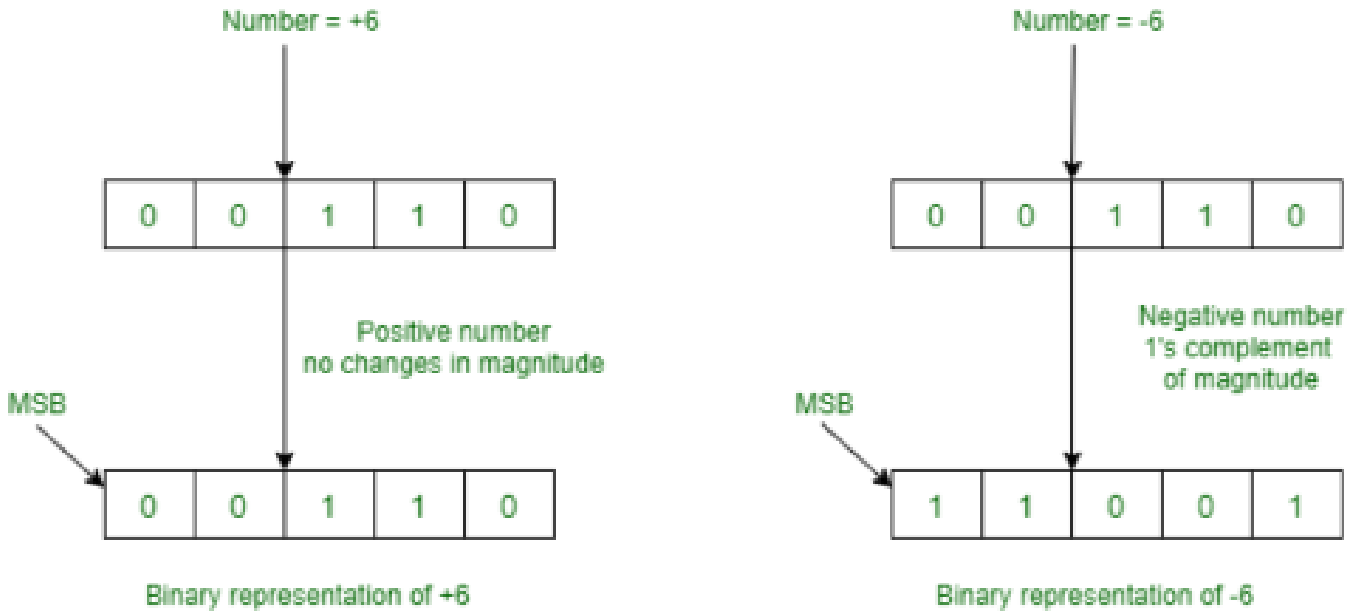
**+ 60 = 0 0 1 1 1 1 0 0**

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

**- 60 = 1 1 0 0 0 0 1 1** (1’s complement of + 60)

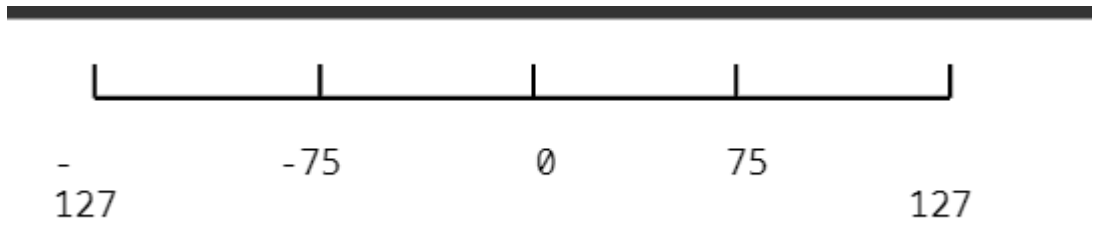
# Example

- Write the one’s complement of 6



# One’s Complement

- What is a complement ?
- It is the opposite of something.
- Because computers do not like to subtract, this method finds the complement of a positive number and then addition can take place.

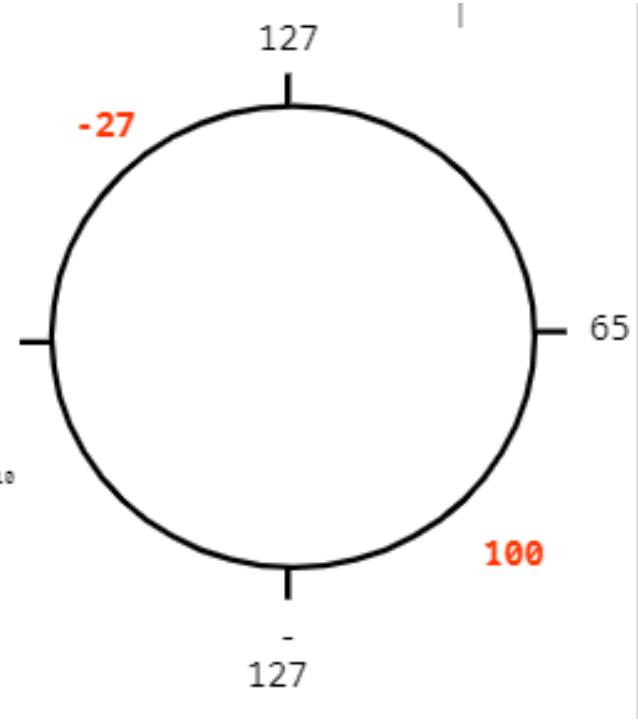


# One’s Complement

- Example Find the one’s complement of +100
- 1. Convert +100 to binary
- .2. Swap all the bits.
- 3. Check answer by adding  127 to your result.

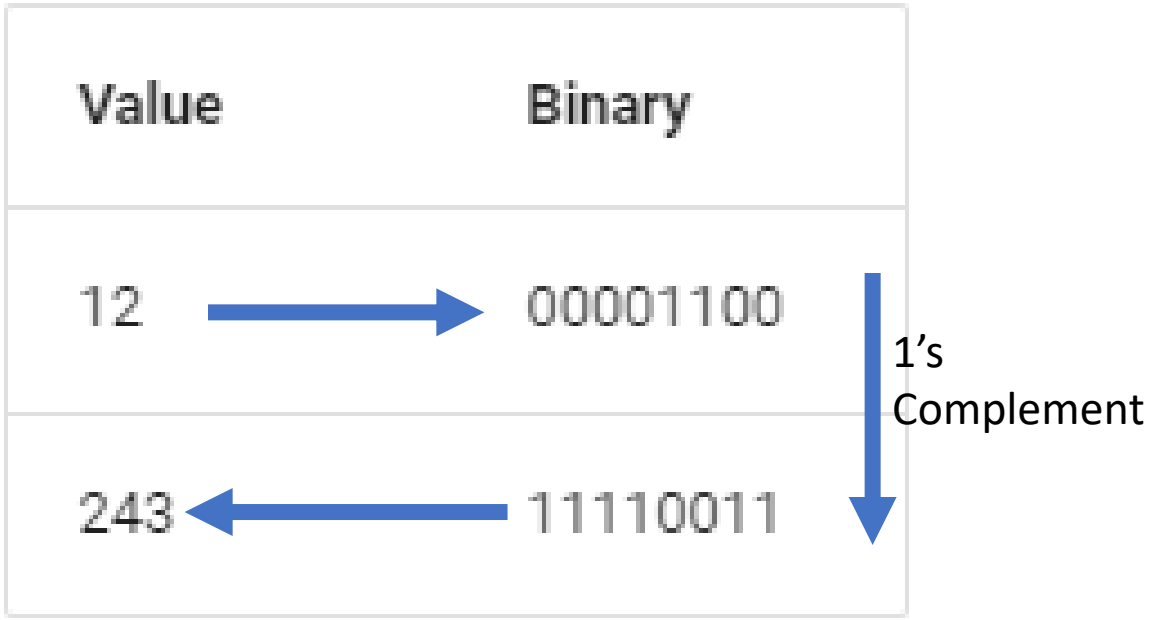
**Example**

1. $+100 = 01100100_2$
2. $\begin{array}{r} 01100100_2 \\ \hline 10011011_2 \end{array}$
3. $10011011_2 = -27_{10}$
4. $127_{10} + -27_{10} = 100_{10}$



# Negative -1s Complement representation

- We can represent 1s complement using the following formula:
- $-x = 2^n - x - 1$
- Consider a number 12.
- If we are using 8 bits 1s complement number, we can represent -12 as -the binary representation of $-12 = 2^8 - 12 - 1 = 243$.



# Two’s Complement

- How can negative numbers be represented using only binary 0’s and 1’s so that a computer can “read” them accurately?
- Consider the binary numbers from 0000 to 1111 (i.e., 0 to 15 in base ten)
- and,
- 1001 to 1111 will represent the negative numbers -1→ -7 respectfully.
- In a computer, numbers are stored in registers where there is reserved a designated number of bits for the storage of numbers in binary form.
- It is easy to change a negative integer in base ten into binary form using the method of two’s complement.

# Two's Complement

Step 1: Write the absolute value of the given number in binary form. Prefix this number with 0 indicate that it is positive.

Step 2: Take the complement of each bit by changing zeroes to ones and ones to zero.

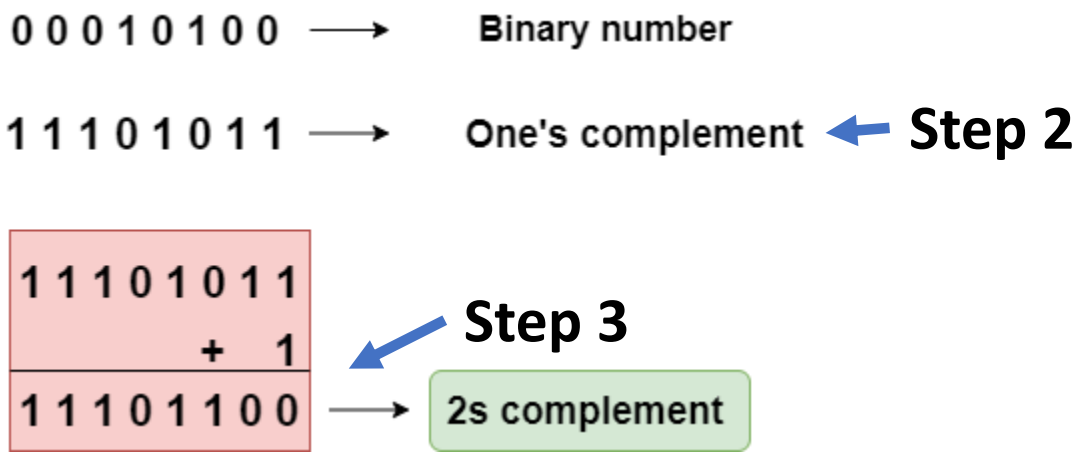
Step 3: Add 1 to your result.

Output is the two's complement representation of the negative integer.

- EXAMPLE:

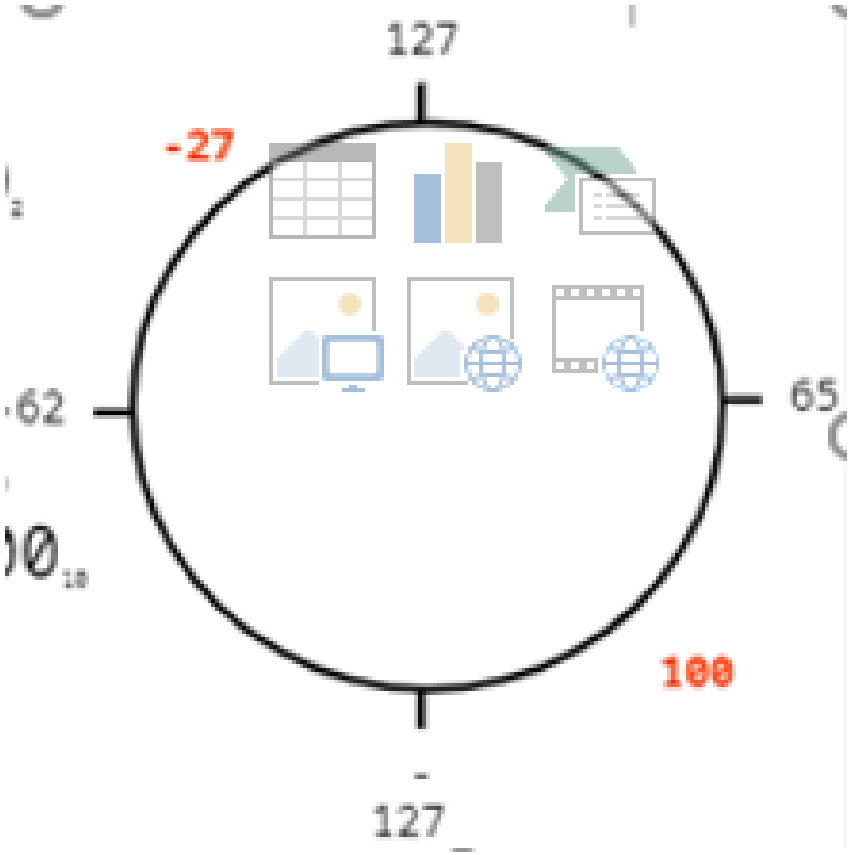
Find 2's complement of -20

Step 1: $20_{10}=00010100_2$



# Two’s Complement

- 8 bits two’s complement
- Find 2’s complement of -63
- Step 1:Absolute value of Binary with prefixes -63= 00111111
- Step 2:take one’s complement, 11000000
- Step 3:Add 1
- Answer 11000001
-



# Floating point numbers

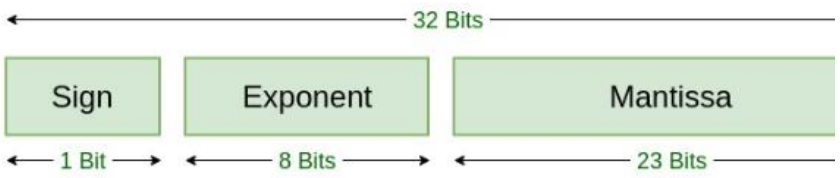
- A floating point number, is a positive or negative whole number with a decimal point.
- For example, 5.5, 0.25, and -103.342 are all floating point numbers
- There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases

# Floating point numbers

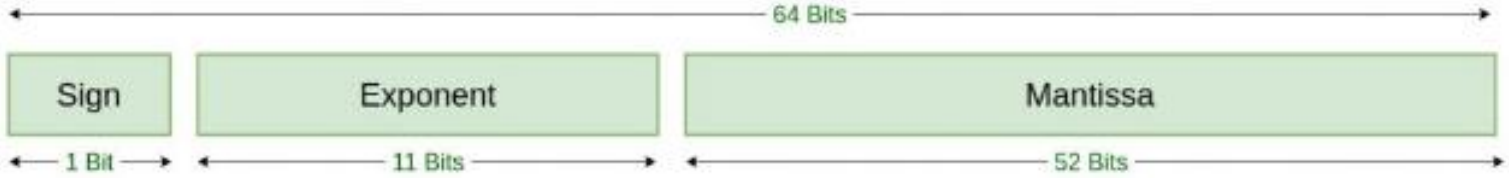
- . IEEE 754 has 3 basic components:
- **The Sign of Mantissa –**
  0 represents a positive number while 1 represents a negative number.
- **The Biased exponent –**
  The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
- **The Normalised Mantissa –**
  The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits.
- Here we have only 2 digits,
- i.e. O and 1.
- So a normalized mantissa is one with only one 1 to the left of the decimal.

# Floating point numbers

- **IEEE 754 numbers are divided into two based on the above three components: single precision and double precision.**




Single Precision
IEEE 754 Floating-Point Standard




Double Precision
IEEE 754 Floating-Point Standard

| TYPES | SIGN | BIASED EXPONENT | NORMALISED MANTISA | BIAS |
|---|---|---|---|---|
| Single precision | 1(31st bit) | 8(30-23) | 23(22-0) | 127 |
| Double precision | 1(63rd bit) | 11(62-52) | 52(51-0) | 1023 |

# Floating point numbers

- Example 1
- 85.125
- 85=101001
- 0.125=001
- 85.125=1010101.001
- =1.0101001x2^6
- Sign=0