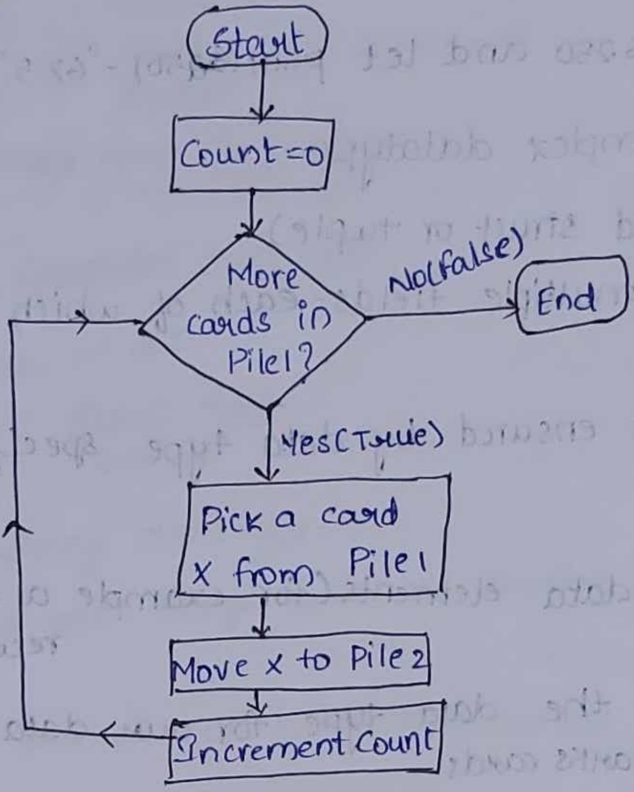
Week-2

Flowcharts

Pictorial representation of computational process

Eg: counting the number of cards



→ Node types.

① Process (Indicated in Rectangular box)

② Decision (Indicated in Diamond shape box)

③ Terminal (Used for start and end. Indicated by ⬭)

→ Arrows indicate operation flow

Pros and cons of flowcharts

Advantages:

- Visual representation of computation
- Easy to understand

Disadvantages:

- Size: Complex processes generate large flowcharts
- Collaboration: Sharing pictures in editable format
- Versions: Compare changes between flowcharts

From pictures to text.

Decribe the previous process in words

Step 0 Start

Step 1 Initialize Count to 0

Step 2 Check cards in Pile 1

Step 3 If no more cards, go step 8

Step 4 Pick a card X from Pile 1

Step 5 Move X to Pile 2

Step 6 Increment Count

Step 7 Go back to Step 2

Step 8 End

Programming language

→ Succinct notation for computational processes

→ Better textual representation for Conditional execution

Step 3 If no more cards, go to Step 8

Step 4 Pick a card X from Pile 1

Repeated execution

Step 2 Check cards in Pile 1

⋮

Step 7 Go back to Step 2

Pseudocode

```
Start
Count = 0
while (Pile 1 has more cards){
 Pick a card X from Pile 1
 Move X to Pile 2
 Increment Count
}
End
```

① Assign a value to a variable

② Repeat steps while condition holds

③ Mark start and end of repeated block

Summary

- Flowcharts are easy to read, visual descriptions of procedures
- ... but they are cumbersome, hard to share and edit.
- Writing down steps in a text is an alternative
- Tune the notation to capture standard features
    - Assigning values to variables
    - Conditional execution
    - Repeated execution

Pseudocode : Iteration and filtering

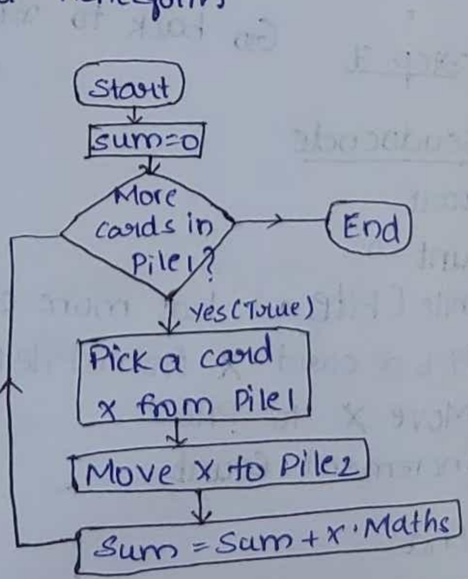
- Counting cards

```
Start
Count = 0
while (Pile 1 has more cards){
     Pick a card x from Pile 1
     Move x to Pile 2
     Increment Count
     }
End
```

- Will dispense with Start and End henceforth

Sum of Maths marks

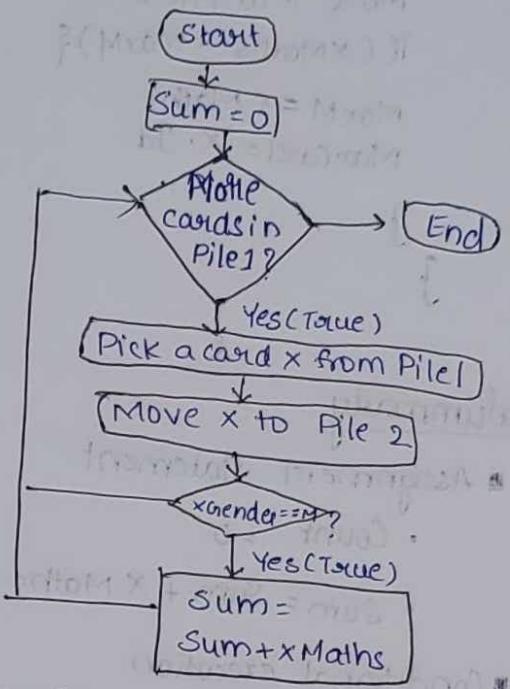
```
Sum = 0
While (Pile 1 has more cards){
Pick a card X from Pile 1
Move X to Pile 2
 Sum = Sum + XMaths
}
```



- Update Sum : assignment statement
  - Sum on right is current value
  - sum on left is updated value
  - = is not mathematical equality
- Increment : Count = Count + 1
- xMaths : Maths marks in card x

Sum of Boys' Maths marks

```
Sum = 0
while (Pile 1 has more cards){
Pick a card x from Pile 1
Move X to Pile 2
 if (xGender = = M){
   Sum = Sum + x Maths
  }
 }
```

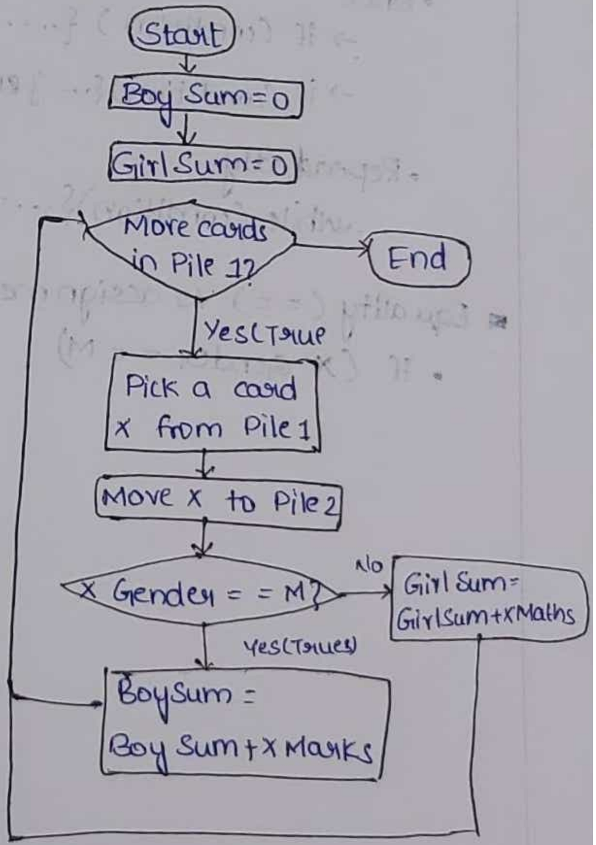


- Conditional execution once.
- Equality (= =) vs assignment (=)

Sum of Boys' and Girls' Maths marks

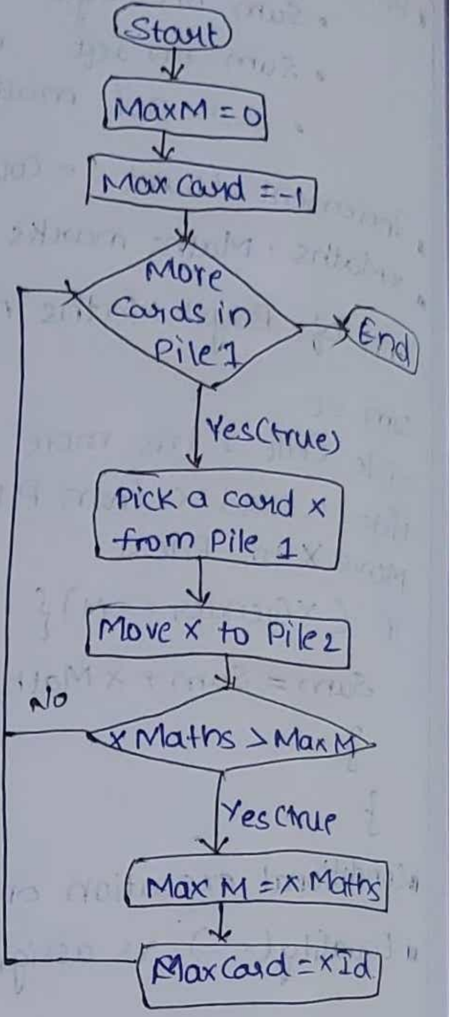
```
BoySum = 0
Girl Sum = 0
While (Pile 1 has more cards){
    Pick a card X
    Move x to Pile 2
    if ( Gender = = M){
     BoySum = Boysum + x Maths
     }
     else {
      Girlsum = Girlsum + x Maths
      }
     }
```

- Alternative branch for conditional



Finding the card with maximum Maths marks

```
MaxM = 0
Max Card = -1
While (Pile 1 has more cards){
    Pick a card X from Pile 1
    Move X to Pile 2
    if (X Maths > MaxM){
    MaxM = X Maths
    Max Card = X. Id
    }
}
```



Summary

- Assignment statement
  - Count = 0
  - Sum = Sum + X Maths.
- Conditional execution
  - Once
    - → if (condition) {. . . }
    - → if (condition) {. . . } else { . . . }
  - Repeatedly
    - → while (condition){. . . }
- Equality (==) vs assignment(=)
  - if (X Gender == M)