Todays Content

1. Class & Object Basics
2. Linked list Intro
3. Create Linked List 1.. N
4. Search Linked List
5. Size of Linked List

Class: It is a blueprint

Object: Instance/Creation of class
: A single class can have multiple objects.

Class → Attributes/Variables: Property of class
Class → Functions/Methods: Behavior of class

```
class Car{
    Inf: Attributes
    string colour;
    string brand;
    int price;
    Functions:
    acceleration()
    ACC()
    music()
};
```

Car : Suhana
colour : red
brand : BMW
price : 1cr
Functions:
acceleration()
ACC()
music()

Car : Shaquib
colour : Black
brand : Ferrari
price : 5Cr
Functions:
acceleration()
ACC()
music()

Note:
1. Each Object will have it's own copy of attributes
2. Function created in class are shared across attributes

Note:
1. Class can have only attributes
2. Class can have only functions
3. Class can have only Attributes / Functions

class creation in C++

```
class student{
  public: # By default all members in class are private;
    int m1, m2;
    int total(){
      return m1+m2;
    }
    student(){ } #
};
```

Default Constructor added.
If there is no constructor by user side

#; missed at end in C++

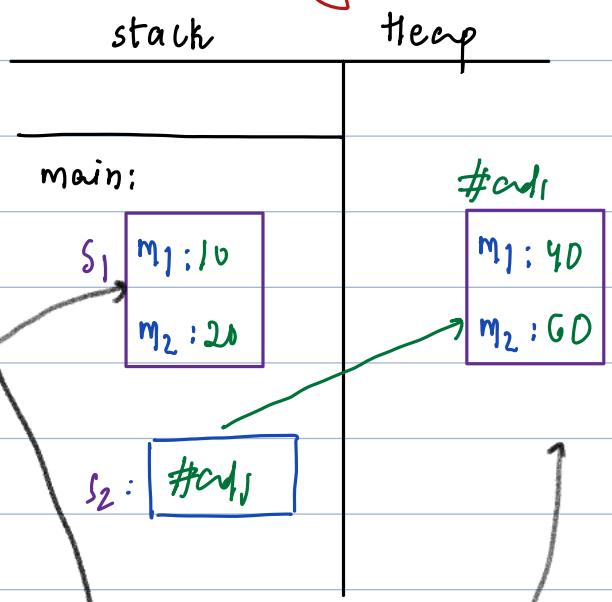
```
main(){
  #Obj:1
    student s1;
    s1.m1=10;
    s1.m2=20
    int x= s1.total();
    print(x); #30

  #Obj:2
    student *s2 = new student();
```

Only pointer will create
reference created Object

```
    s2->m1 = 40
    s2->m2 = 60
```

↳ # Arrow operator to access member of class by pointer

```
    printf( s2 -> total()); #100
```

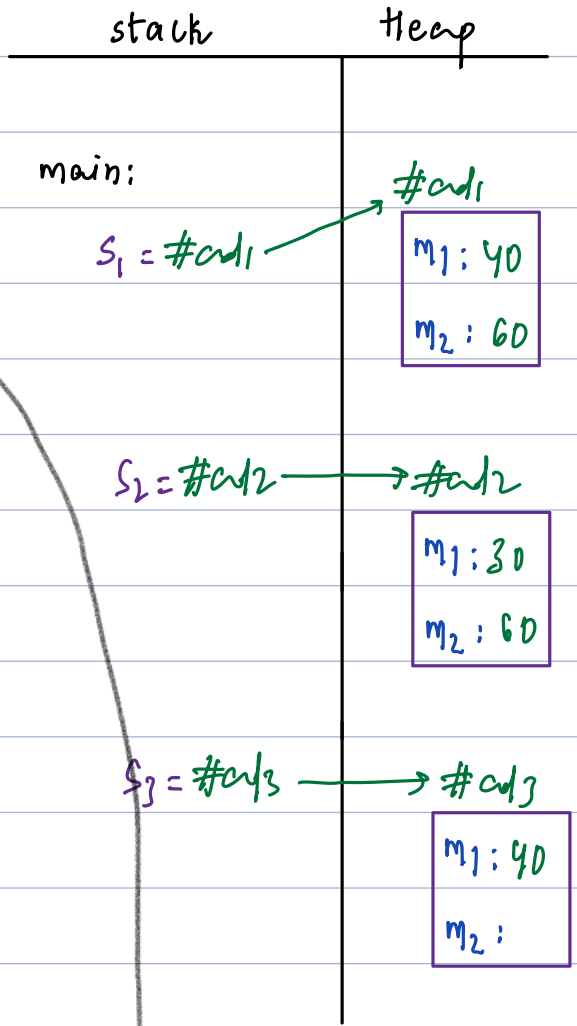


Constructor In Class:

1. Just like function, no return type & name same as class name
2. We can only call constructor which exist.

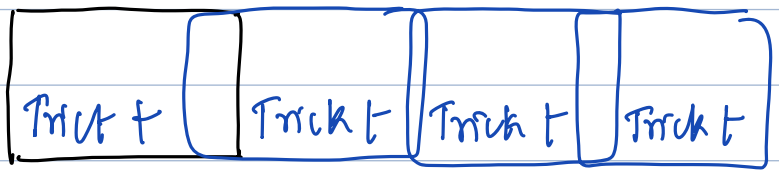
Note: If we add constructor, default constructor will not be added.

```
class student{
    public:
        int m1, m2;
        student(int x, int y){
            m1 = x;
            m2 = y;
        }
        student(int x){
            m1 = x;
        }
};

main(){
    student *s1 = new student();
    s1->m1 = 40;
    s1->m2 = 60;

    student *s2 = new student(30, 60);

    student *s3 = new student(40);

    student *s4 = new student(); # Error.
}
```



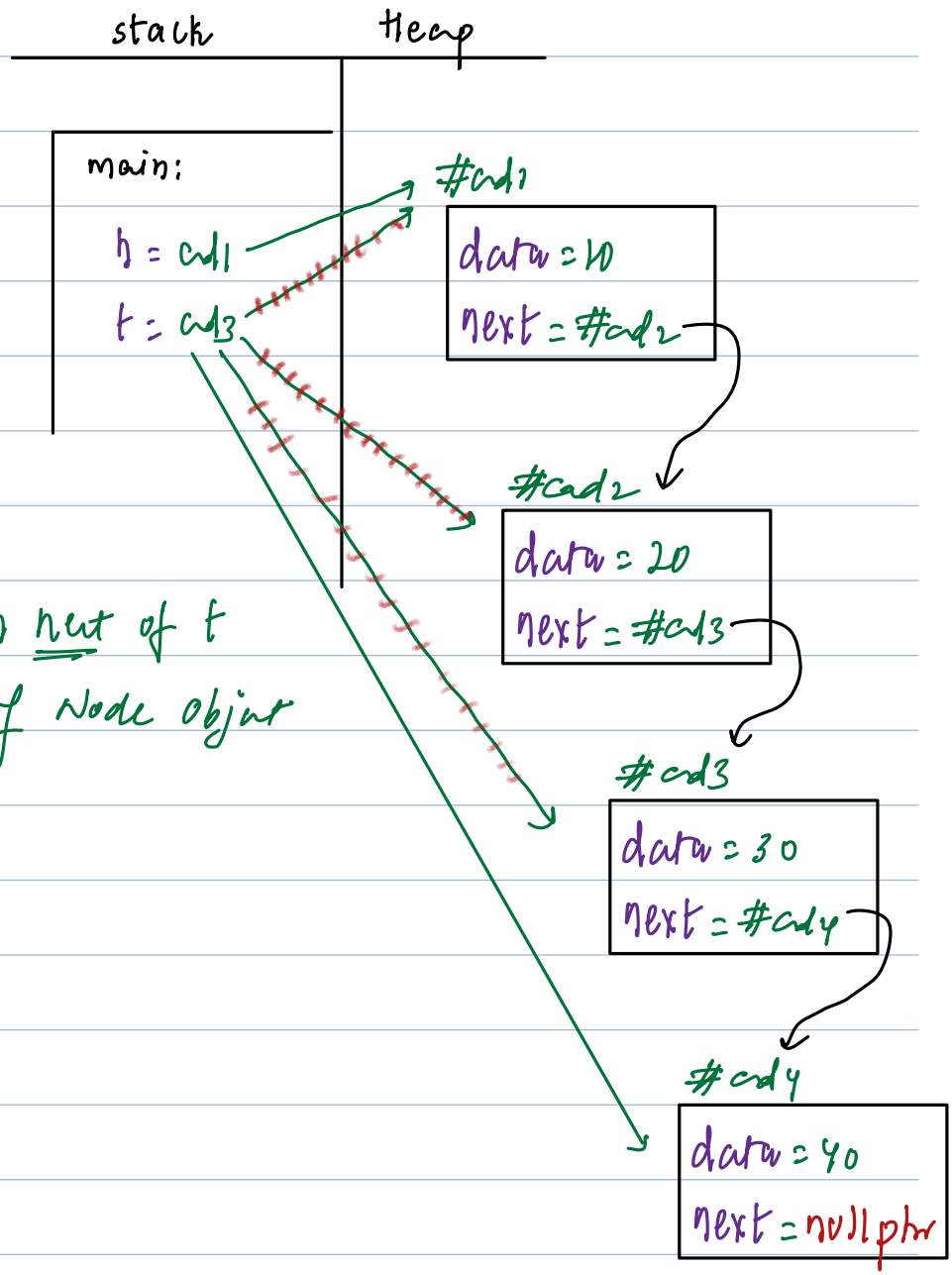
No suitable for Data structure like Linked List / Trees / - — ..

```
class trick {
  public:
    trick t;
};
main() {
  trick a;
```



Object reference as member.

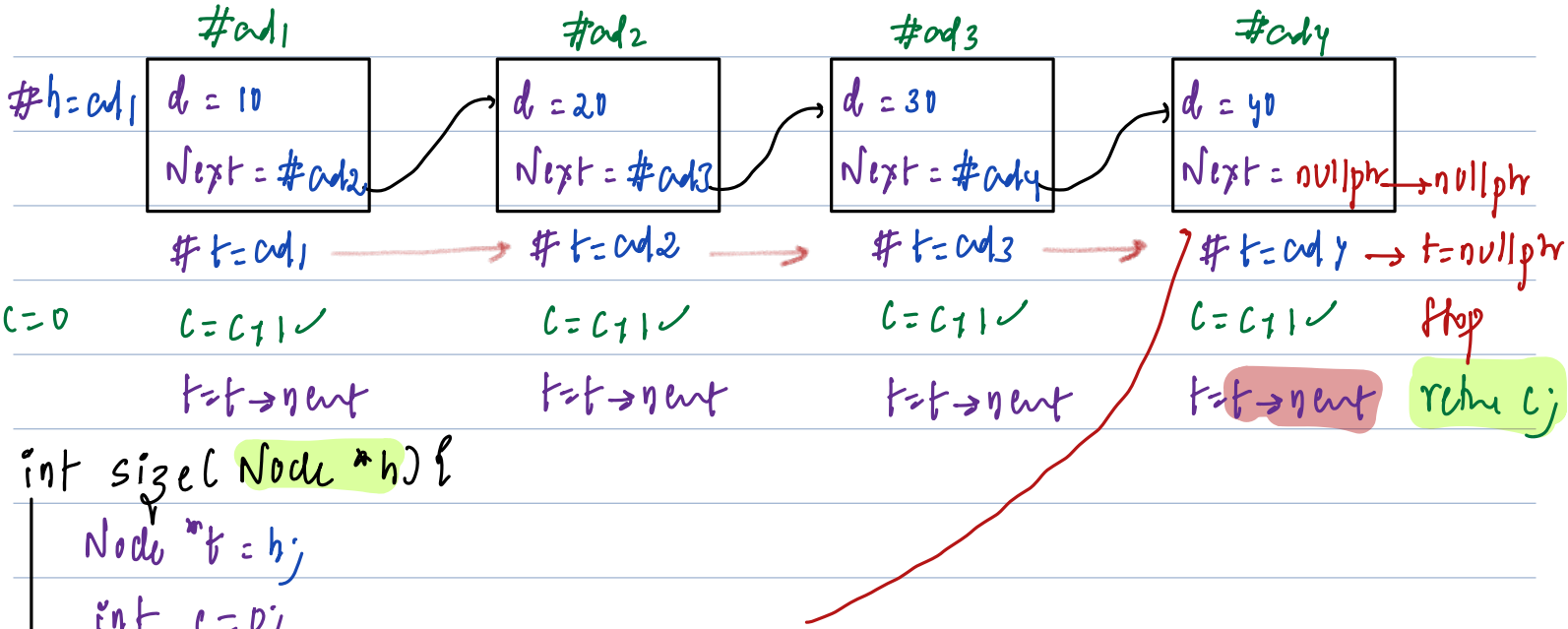
```
class Node{
  public:
    int data;
    Node *next;
    Node(int n){
      data = n
    } next = nullptr; # In Java = NULL;
};
main(){
  Node *h = new Node(10);
  print(h); #add1;

  Node *t = h;

  t->next = new Node(20);
        Address is stored in next of t
    Node* can store address of Node object
  t = t->next;

  t->next = new Node(30);
  t = t->next;

  t->next = new Node(40);
  t = t->next;
}
```



#Obs1: Advantage of nullptr it will indicate reach of end

#Obs2: If we have head node address we can traverse entire linked list

18 Given a head Node of a linked list, return size of it.




```
int size(Node *h){
    Node *t = h;
    int c=0;
    while(t!=nullptr){
        c=c+1;
        t=t->next
    }
    return c;
}
```

Code2 # Will not count last node

```
Node *t = h;
int c=0;
while(t->next!= nullptr){
    c=c+1;
    t=t->next
}
return c;
```

→ If t=nullptr
nullptr->next != nullptr;
Fails RTE.

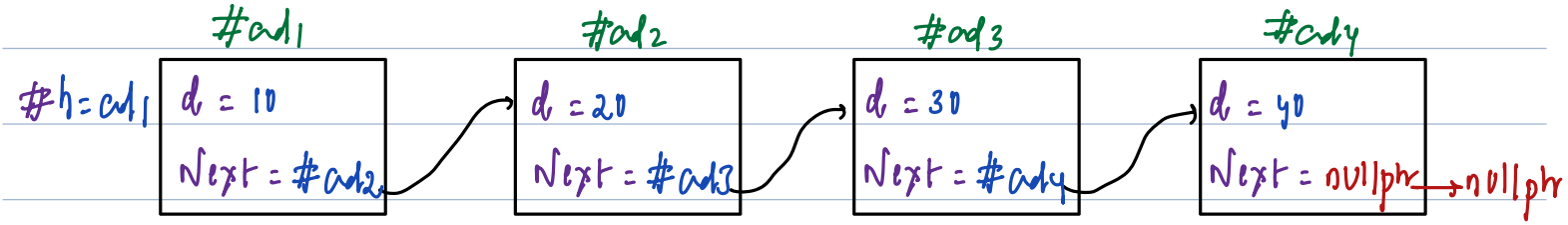
Code3

Eg: h = nullptr;

```
Node *t = h;   # t = nullptr
int c = 1;     # c = 1
while( t->next != nullptr){   # nullptr->next != nullptr;
                              # Fails RTE.
    c = c+1;
    t = t->next
}
return c;
```

Make sure t != nullptr

Before we access t->data or t->next;




```
t = add1;
    print(t)          #add1
    print(t->next)    #add2  # if won't effect t;
    print(t->next->next);   #add3
    print(t->next->next->data);   #30
```

if(t != nullptr && t->next != nullptr)

Before we access t->next->data or t->next->next;