* Hash tables

1) Hash table:-

- Hash tables is a data structure used for storing and retrieving data quickly.
- Every entry in hash table is made using Hash function.

2) Hash function:-

- Hash function is a function used to place data in a hash table.
- Similarly hash function is used to retrieve data from hash table.

1. Linear Probing

- When collision occurs i.e. when two records demand for the same location in the hash table, then the collision can be solved by placing second record linearly down wherever the empty location is found.

e.g; m = 10 keys = {131, 4, 5, 7, 8, 21, 31, 61}

	Index	data
	0	
131 % 10 = 1	1	131
21 % 10 = 1	2	21
31 % 10 = 1	3	31
4 % 10 = 4	4	4
5 % 10 = 5	5	5
61 % 10 = 1	6	61
7 % 10 = 7	7	7
8 % 10 = 8	8	8
	9	

2. Double hashing

- Double hashing is a technique on which a second hash function is applied to the key when a collision occurs.
- By applying the second hash function we will get the number of positions from the point of collision to insert.

Q. Insert following keys into hash table using quadratic probing where table size m=7 and h1(x) = x mod m, h2(x) = 5 - (x mod 5), key = { 76, 93, 40, 48, 5, 55 }

insert 76
76%7 = 6

0	
1	
2	
3	
4	
5	
6	76

probes: 1

insert 93
93%7 = 2

0	
1	
2	93
3	
4	
5	
6	76

probes: 1

insert 40
40%7 = 5

0	
1	
2	93
3	
4	
5	40
6	76

probes: 1

insert 47
47%7 = 5
5-(47%7) = 3

0	
1	47
2	93
3	
4	
5	40
6	76

probes: 2

insert 10
10%7 = 3

0	
1	47
2	93
3	10
4	
5	40
6	76

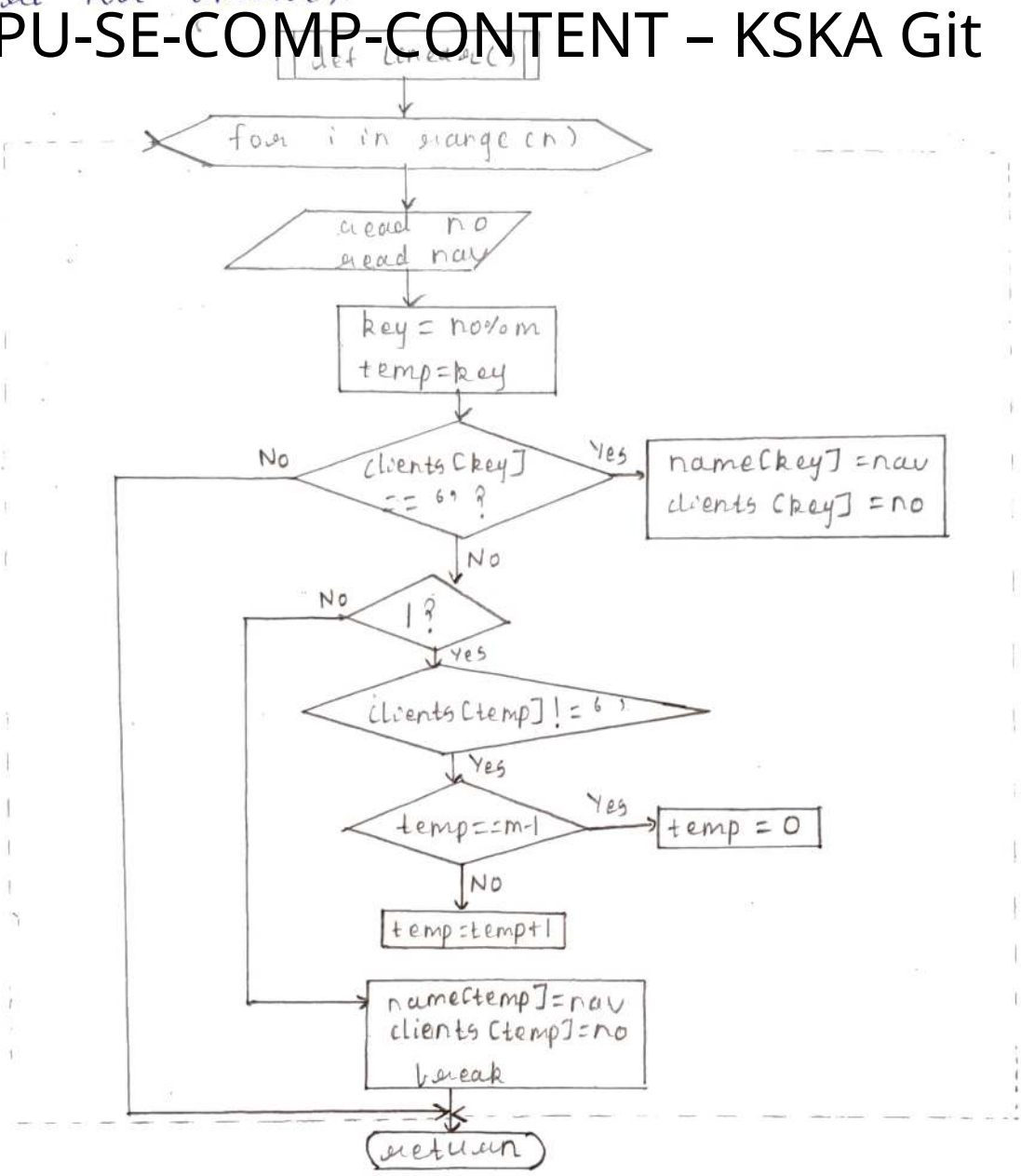
probes: 1

insert 55
55%7 = 6
5-(55%5) = 5

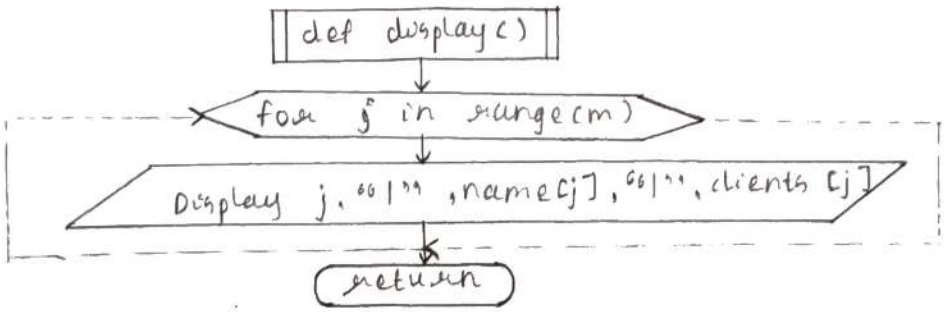
0	
1	47
2	93
3	10
4	55
5	40
6	76

probes: 2

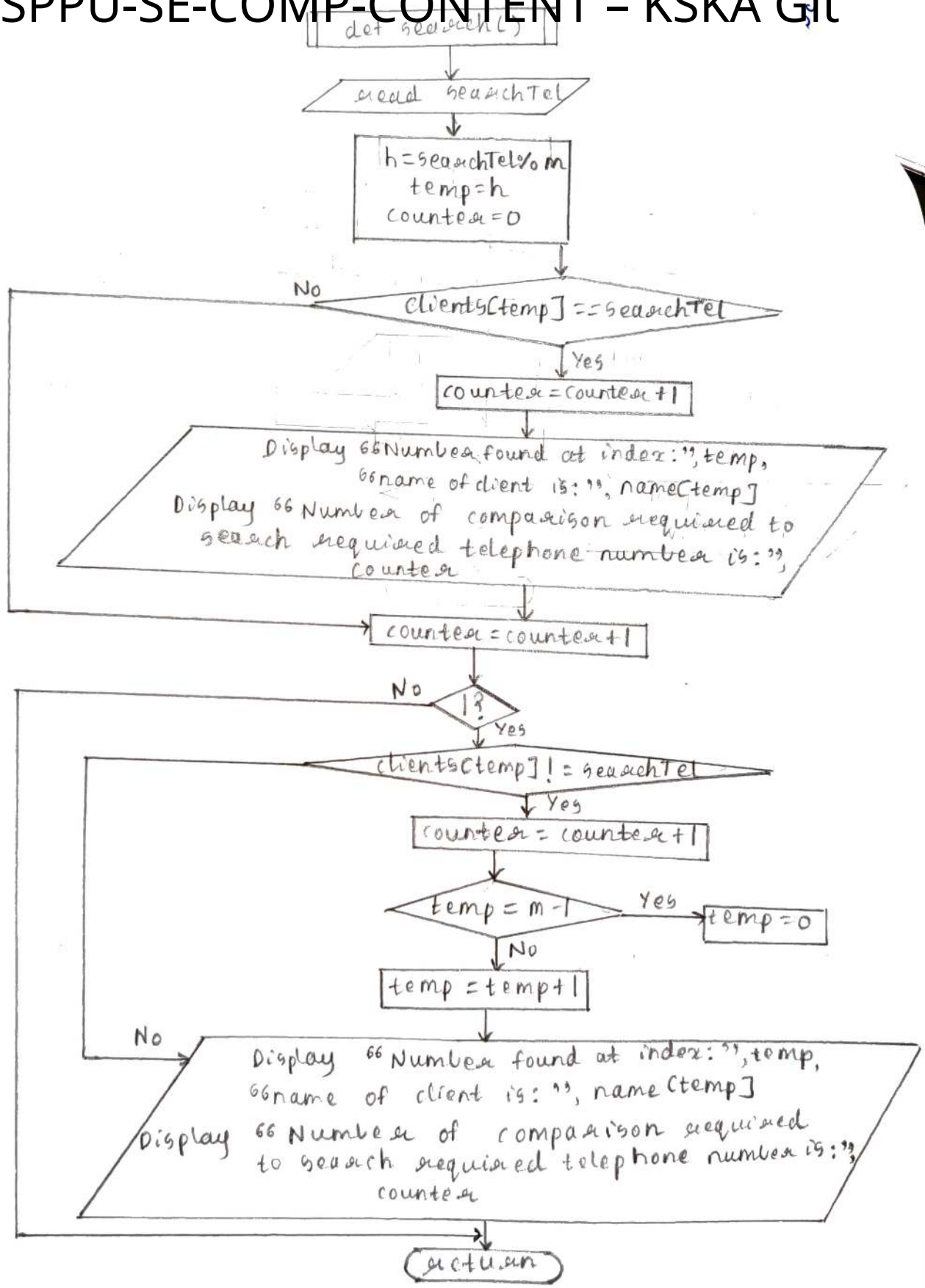
owchart for linear().

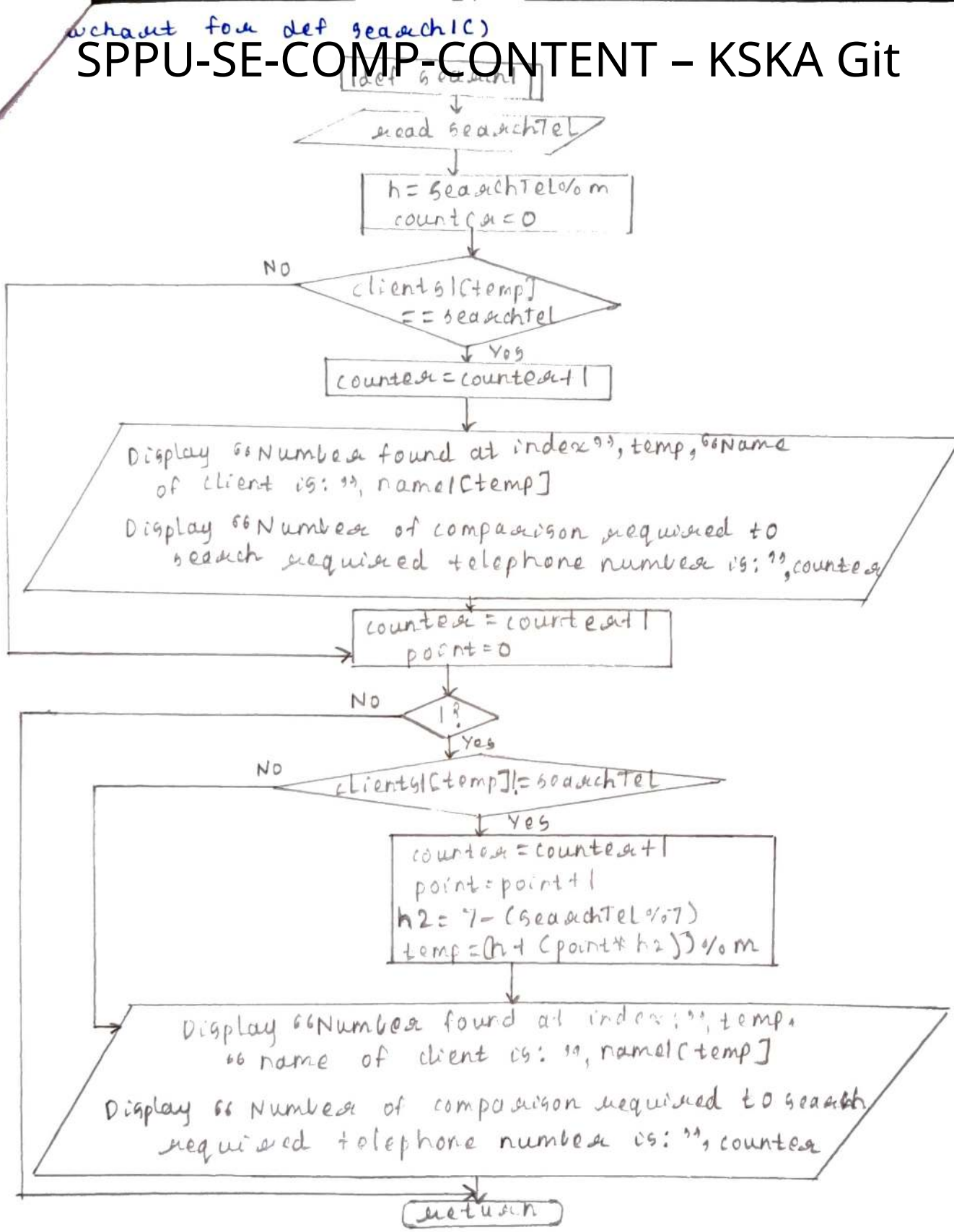


→ Flowchart for display()

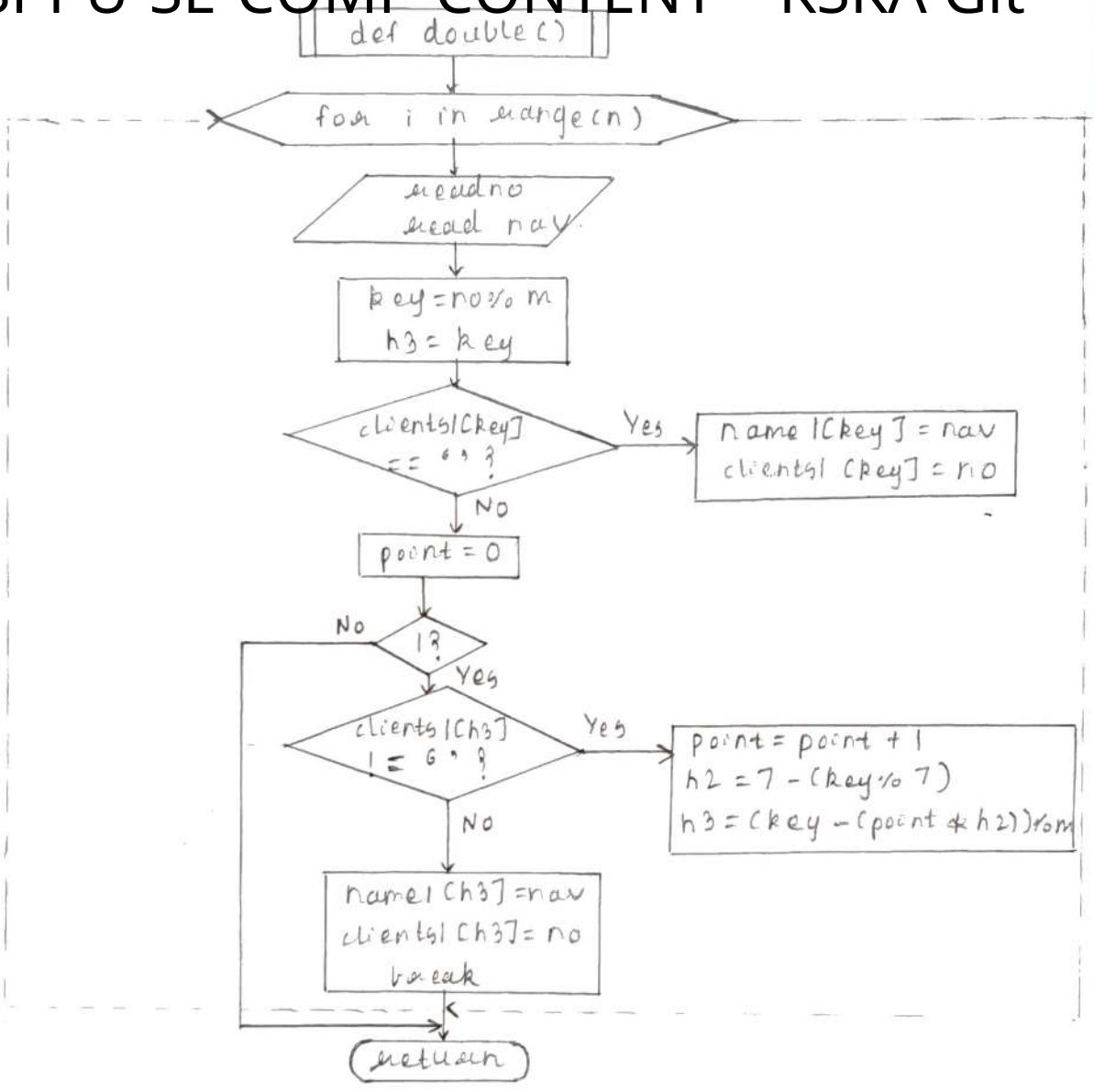


→ Flowchart for search()

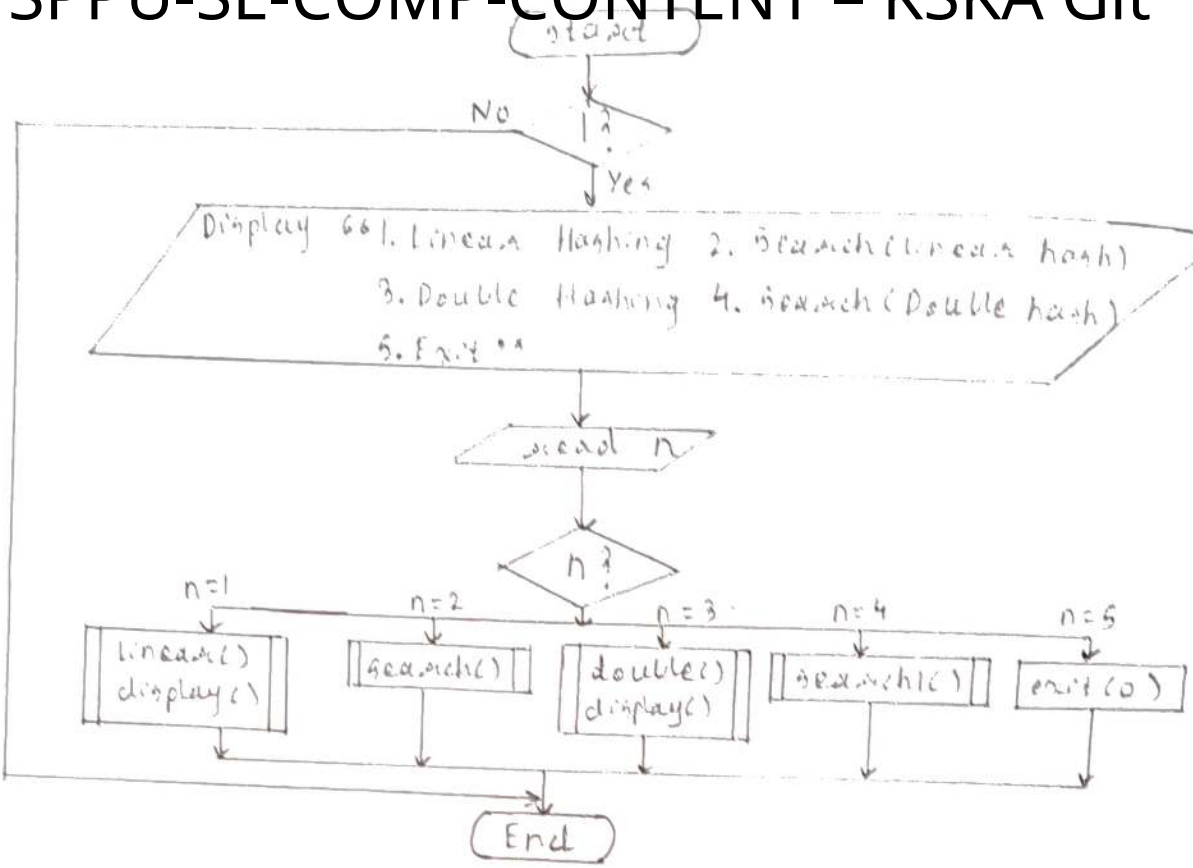




Flowchart for double()



→ Flowchart for main()



→ Pseudocode for linear()

```
1. for i in range(n) do
   begin
       readd no
       read nav
       store no%m on key
       store key in temp
       if clients[key] == '' then
           store nav in name[key]
           store no in clients[key]
       else
           while(1) do
           begin
               if clients[temp] != '' then
                   if temp == m-1 then
                       temp = 0
                   else
                       increment temp
               else
                   store nav in name[temp]
                   store no in clients[temp]
                   break
   end
2. return
```

→ Pseudocode for display()

```
1. for j in range(m) do
   begin
       Display j, "|", name[j], "|", clients[j]
   end
2. return
```

→ Pseudocode for search()

```
1. read searchTel
2. initialize check = False
3. for c in range(m) do
   begin
        if name[c] == searchTel then
              Display "Telephone number: ", clients[c]
              check = True
   end
4. if check == False then
        Display "Enter valid name!!"
5. return
```

→ Pseudocode for double()

```
1. for i in range(n) do
   begin
        read no
        read nav
        store no%m in key
        store key in h3
        if clients1[key] == 0 then
              Declare name1[key] = nav
                      clients1[key] = no
        else
              initialize point = 0
              while(1) do
              begin
                    if clients1[h3] != 0 then
                          increament point
                          initialize h2 = 7 - (key % 7)
                          initialize h3 = (key + (point * h2)) % m
                    else
```

→ Pseudocode for def search

```
1. read searchTel
2. calculate h = searchTel % m
3. store h in temp
4. initialize counter = 0
5. if clients[temp] == searchTel then
        increment counter
        Display "Number found at index: ", temp,
            "name of client is: ", name[temp]
        Display "Number of comparisons required
            to search required telephone number
            is: ", counter
   else:
        increment counter
        while (1) do
        begin
            if clients[temp] != searchTel then
                increment counter
                if temp == m-1 then
                    initialize temp = 0
                else:
                    increment temp
            else:
                Display "Number found at index: ",
                temp, "name of client is: ", name[temp]
                Display "Number of comparisons
                required to search required
                telephone number is: ", counter
                break
        end
6. return
```

→ Pseudocode for def search1()

```
1. read searchTel
2. calculate h = searchTel % m
3. store h in temp
4. initialize counter = 0
5. if clients1[temp] == searchTel then
       increment counter
       Display "Number found at index: ", temp, "name
           of client is ", name1[temp]
       Display "Number of comparison required to
           search required telephone number is:",
           counter
   else:
       increment counter
       initialize point=0
       while() do
       begin
           if clients1[temp] != searchTel then
               increment counter
               increment poont
               calculate h2 = 7 - (searchTel % 7)
               calculate temp = (ht (poont * h2)) % m
           else:
               Display "Number found at index: ",
               temp, "name of client is: ", name1[temp]
               Display "Number of comparisons
               required to search telephone
               number is: ", counter
               break
       end
6. return
```

```
                    Declare name1[ch3] = nav
                    clients1[ch3] = no
                    break
                end
end
2. return

→ Pseudocode for main()
1. ~~while(1)~~ Start
2. while (1) do
begin
    Display "1. Linear Hashing 2. Search (Linear hash)
             3. Double Hashing 4. Search (Double hash)
             5. Exit "
    read n
    if (n==1) then
        call function Linear()
        call function display()
    elif n==2 then
        call function search()
    elif n==3 then
        call function double()
        call function display1()
    elif n==4 then
        call function search1()
    elif n==5
        exit(0)
    elif n<0 and n>5
        Display "Enter valid choice !!! "
end
3. End
```

Q1. Explain different hashing functions with example.

Ans. 1. Division method :-

→ Idea :

- Computes hash value from key using the % operator
- Map a key k into one of the m slots by taking the remainder of k divided by m.

$$h(k) = k \bmod m$$

→ Example :-

- $k = 1276$, $n = 10$

$$h(1276) = 1276 \bmod 10 = 6$$

2. Multiplication method :-

→ Idea :

- Multiply key k by a constant A, where $0 < A < 1$
- Extract the fractional part of kA and multiply the fractional part by m
- Take the floor of the result

$$h(k) = \lfloor m (kA \bmod 1) \rfloor$$

→ Example :-

$k = 123$, $m = 100$, $A = 0.618033$

$h(123) = 100 (123 \times 0.618033 \bmod 1)$

$= 100 (76.018059 \bmod 1)$

$= 100 (0.018059) = 1$

3. Digit Extraction method :-

→ Idea :

- Selected digits are extracted from the key and used as address

Address = Selected digits from key

→ Examples -

- If six digit employee number is 379245 then select first digit as the index so 379 is the key address.

4. Folding :-

→ Idea:

- It involves splitting keys into two or more parts and then combining the parts to form the hash addresses.

→ Example:

- To map the key 25936715 to a range between 0 and 9999, we can:

i) split the number into two as 2593 and 6715 and

ii) add these two to obtain 9308 as the hash value.

5. Mid-square method :-

→ Idea:

- The key is squared and the middle part of the result taken as the hash value.

→ Example:

- To map the key 3121 into a hash table of size 1000, we square it $3121^2 = 9740642$ and extract 406 as the hash value.

Q2. Describe extensible hashing for the given input keys: 1, 10, 7, 8, 15, 16

Ans. Elements :- 1, 10, 7, 8, 15, 16

Bucket size :- 2 (Assume)

1 → 00001

10 → 01010

7 → 00111

8 → 01000

15 → 01111

16 → 10000

→ For Directory 1,

$2^1 = 2$

1	1
0	1, 7, 15 → overflow
1	

→ For Directory 2,

$2^2 = 4$

2	2
00	8, 16
01	2
10	1
11	2
	10
	2
	7, 15