Assignment No. 7.

Title :- Symbol table

problem statement :- The symbol table is generated by compiler. from this perspective, the symbol table is set of name-attribute pairs. In symbol table for compiler, the name is an identifier, and attribute might include an initial value & a list of lines that use the identifier, perform following operation.

a) Determine if particular name is in table.
b) Retrieve the attribute of that name.
c) Modify the attribute of that name.
d) Insert a new name & its attribute
e) Delete a name & attribute

Objective :- i) To learn & implement the linear probing with chaining
ii) To learn & implement the linear probing with & without replacement.

Theory :-

Symbol table :- It is DS used by the compiler, where each identifier in program's source code is stored along with information associated with it relating to its declaration.

ymbol table can be implemented through following DS:
i) Linked list
ii) Hashtable
iii) Tree

- Collision resolution technique:-

During inseetion of the new item into-hash table, the sequence of location that we examine called probe sequence.

- lineae probing with replacement with chaining & without replacement

It is simplee to implement. In chaining, Hash table nevee fills up, we can always add more element to chain. chaining is mostly used when it is unknown how many & how frequently keys may be inseeted oe deleted.

e.g. Inseet following sequence lineae probing with chaining (with & without replacement)

RBI, RY, KTM, SSS, ARD, ARG, AGP, EM, GVK, SAT, MST, RK

chaining with replacement

hash	Identifiee	chain	hash	Identifiee	chain
0	ARD	1	6	gvk	-1
1	ARG	2	5		
2	AGP	-1	7		
3			8		
4	em	-1	9		
5			10	KTM	-1

hash	Identifiee	chain	hash	Identifiee	chain
11			18	SSS	20
12			19	Ry	21
13			20	SAJ	-1
14			21	RK	-1
15			22		
16			23		
17	RBJ	19	24		
			25		

chaining without replacement

hash	Identifiee	chain	hash	Identifiee	chain
0	ARD	1	13		
1	ARG	2	14		
2	AGP	-1	15		
3			16		
4	em	-1	17	RBJ	18
5			18	RY	21
6	gvK	-1	19	SSS	20
7			20	SAT	-1
8			21	RK	-1
9			22		
10	KTM	-1	23		
11			24		
12	MST	-1	25		

Pseudocode :-

- Insertion :- (without replacement)

```
step 1:) Read the key identifiee & its attributes (scope oe type)
step 2:) calculate the hash value using hashfunc() say hashaddr.
step 3.) if ( hashtable [hashaddr] is empty ) then
                store the key & value at hashaddr set chain as
                -1.
         else
            3.1) set one pointee to next index of hash addr
                 through chain
                 while (chain != -1)
                   {
                      go to the through the chain
                   }
            3.2) if current chain pointee is Null then
                   store the key & value with attributes.
                 ~~else~~
Step 4:) END
```

Inseetion :- (with replacement)

```
Step 1:) Read the key identifiee & its attribute
step 2:) calculate hash value using hashfunc() say hasha
Step 3:) if (hashtable [hashaddr] is empty) then
            store the key with attributes
```

```
else.
  3.1) Set one pointee to next index of hash addr
         through chain say hashaddr1
           if (hashtable[hashaddr1] == hashaddr)
               {
               Replace that hashaddress key & value
                 if (Next hash address is empty)
                     {
                     place the key & value of previous hashaddr
                     }
                 else
                     {
                     Search to next empty location
                     }
               }
END
```

Deletion :-

```
Step 1:) Read key value to be deleted say k.
Step 2:) calculate hash value of k say kaddr
Step 3:) if (HashTable[kaddr] == k)
                 Delete the key with attributes
         else
           go to the next through chain
             while (chain != -1 && HashTable[hashaddr] != k
                 {
                 go to transverse Chain
                 }
```

```
        if ( HashTable[current chain] == k)
                delete Key & value associated with it.

Step 4:) END
```

• Searching :-

```
Step 1:) Read the key which is to be searched. say k
Step 2:) calculate hash value of key say hashaddr
Step 3:) set pointer say temp to head [index]
         while (temp ! = NULL)
         {
               if (temp -> Key == k )
               {
                     print attributs
                     return;
               }
             temp = temp -> next
         }
Step 4:) return 0;
```

Conclusion :- Linear probing with chaining, hash table never fills we can add more elements to chain. chaining is mostly used when it is unknown how many & how frequently keys may be inserted or deleted. & we have implement the Linear probing with chaining.