N o. EXPERIMENT : _____

PG. NO.	DATE

Assignment - 7

Problem Statement - The symbol table is generated by the compiler. From this perspective, the symbol table is a set of name-attribute pairs. In a symbol table, for a compiler the name is an identifier, and the attributes might include an initial value & a list of lines that we use the identifier perform.

a) Determine if a particular name is in the table.
b) Retrieve attributes of that name.
c) Modify attributes of that name.
d) Insert a new name & its attributes (Use chaining with & without replacement)

Objective -

(i) To understand concept of symbol table.
(ii) Why symbol table is needed.

Outcome -

(i) Use of symbol table
(ii) Various methods of implementing symbol table.

Theory -

Symbol Table -

(i) It is a data structure used by a language translator such as compiler or interpreter, where each identifier in a program's source code is associated with information relating

to its declaration or appearance in the source code.

(ii) A symbol table may only exist during the translation process, or it may be embedded in the output of the process, such as in an OBJ object file or later.

(iii) It is used to store information related to various entities like function name, variable name, objects, classes, interfaces, etc.

(iv) When the identifiers are found, they will be entered into the table, which will hold the relevant information about identifiers.

(v) It is a type of data structure that captures scope information.

- Symbol Table Implementation Methods -

1) Unordered array
2) Ordered array
3) BST
4) Hashing
5) Balanced Search Trees.

Teacher's Sign : _____

Pseudo code -

(i)
```
void insert (string key)
{   string attribute;
    int hcode = (int) key[0] - 97;
    int i = hcode;
    Enter attribute
    if (name[hcode] == "\0") {
        name[hcode] = key;
        attr[hcode] = attribute;
    }
    else {
        if (name[hcode][0] == key[0]) {
            if (chain[hcode] == -1)
            {
                while (name[i] != "\0")
                    { i++; }
                name[i] = key;
                attr[i] = attribute;
                chain[hcode] = i;
            }
            else {
                int k = i;
                while (chain[k] != -1)
                    { k = chain[k]; }
                i = k; while (name[i] != "\0")
                    { i++; }
                name[i] = key;
                attr[i] = attribute;
                chain[k] = i;
            }
        }
```

N o. EXPERIMENT : _____

PG. NO.	DATE

```
else {
while (name[i] != "\0")
    { i++; }
 name[i] = key;
  attr[i] = attribute;
 }
}
}
```

2.
```
void modify (string key)
{
    string change;
    Enter change;
   int fposi = int(key[0]) - 97
   int i = fposi;
   if (name[fposi] == key)
      attr[fposi] = change;
   else
   {   while (name[i] != key)
       {
          if (name[i] == "\0")
          { Print "Not Found";
            End; }
         else
            i++;
       }
      attr[i] = change;
   }
}
```

Test cases -

Case	Expected Output		Actual Output
	Name	Attribute	
1. Enter key : a Enter attribute : b, 4	0] a	b, 4	Yes
2. Enter key : a Enter attribute : 8, x	0] a 1] a	b, 4 8, x	Yes
3. Enter key : b Enter attribute : f	0] a 1] a 2] b	b, 4 8, x f	Yes
4. Enter key : a Modify : l	0] a 1] a 2] b	l 8, x f	Yes

Conclusion -

Symbol Table was implemented successfully using hashing in C++.