

Compiler Design Lab (CS 306L)

Week 3: Symbol Table Implementation

⇒ **Time Complexity:**

- For Hash Table ⇒ Constant time
- For Linked List ⇒ $O(N)$

⇒ **Space Complexity:** $O(N)$ for both - Hash Table and Linked List

⇒ **Program for LinkedList:**

//static program

```
#include <iostream>
```

```
using namespace std;
```

```
struct Node{  
    string name;  
    struct Node* next;  
    struct dt *para;  
    struct arr *arrayPara;  
} *head = NULL;
```

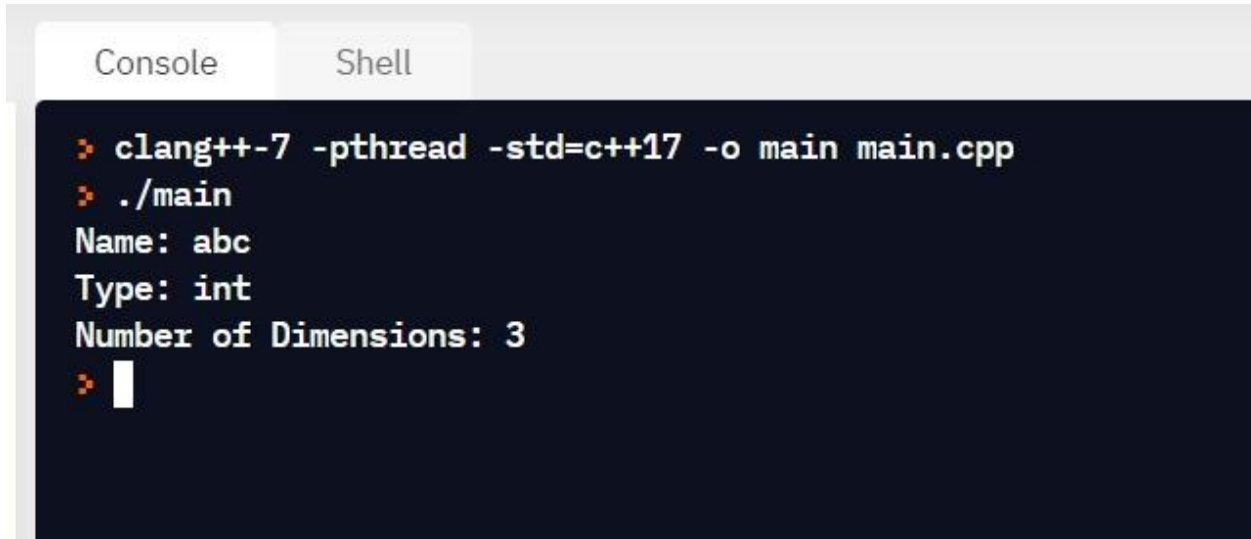
```
struct dt{  
    string type; //for the datatype of the variable  
    string scope; // either global or local  
};
```

```
struct arr{  
    string type; //for the datatype of the variable  
    int noOfDimensions; //whether it is a one dimensional array or 2 dimension or n dimensions  
                        // => eg. a[3][4] is a 2-D array  
    int *dimensions; //and in those the first dimension is 3 and second one is 4  
};
```

```
int main()  
{  
    struct Node* temp = ( struct Node* ) malloc(sizeof(struct Node));  
    struct arr* arrayPointer = ( struct arr* ) malloc(sizeof(struct arr));  
    head = temp;  
    temp -> para = NULL;  
    head -> arrayPara=arrayPointer;  
    head -> name="abc";  
    head -> arrayPara->type="int";  
    head -> arrayPara->noOfDimensions=3;  
    head -> arrayPara->dimensions = (int*)  
    malloc(sizeof(int)*head->arrayPara->noOfDimensions);
```

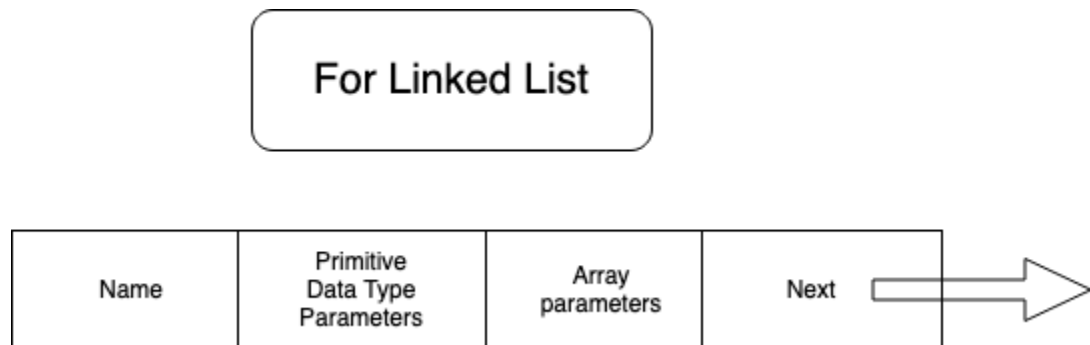
```
cout <<"Name: "<<head->name<<endl;
cout <<"Type: "<<head->arrayPara->type << endl;
cout <<"Number of Dimensions: "<<head->arrayPara->noOfDimensions << endl;
return 0;
}
```

Output Screenshot:



```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
Name: abc
Type: int
Number of Dimensions: 3
> |
```

Working structure:



⇒ **Program for HashTable:**

```
//static program
#include <iostream>
#define MAX_NAME 256
#define TABLE_SIZE 25
using namespace std;

struct Node{
    string name;
    struct dt *para;
    struct arr *arrayPara;
}*head=NULL;

struct dt{
    string type;
    string scope;
};

struct arr{
    string type;
    int noOfDimensions;
    int *dimensions;
};

Node* table[TABLE_SIZE];

//the index values for the given keys in the hash table
int HashFunction(string name)
{
    int length = name.length();
    int hash_value = 0;
    for (int i = 0; i < length; i++)
    {
        hash_value += (name[i]);
    }
    hash_value = hash_value % TABLE_SIZE;
    return hash_value;
}

//initialising all the values in the table as null
void IntiTable()
{
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        table[i] = NULL;
    }
}
```

//insertion of a new node in the table

```
bool InsertInTable(Node *p)
{
    if(p == NULL) return false;
    int index = HashFunction(p -> name);
    if(table[index] != NULL)
    {
        return false;
    }
    table[index] = p;
    return true;
}
```

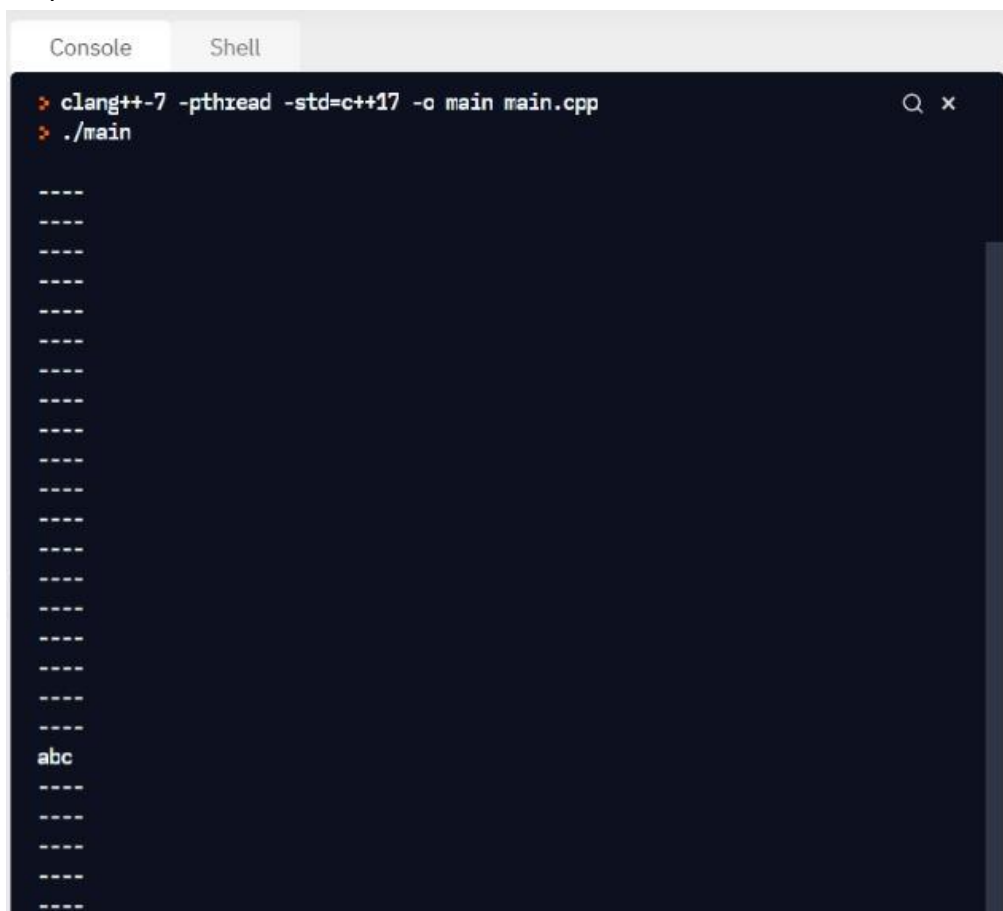
//printing the values in the table

```
void PrintTable()
{
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        if (table[i] == NULL)
        {
            cout << "---- \n";
        }
        else
        {
            cout << table[i] -> name << "\n";
        }
    }
}
```

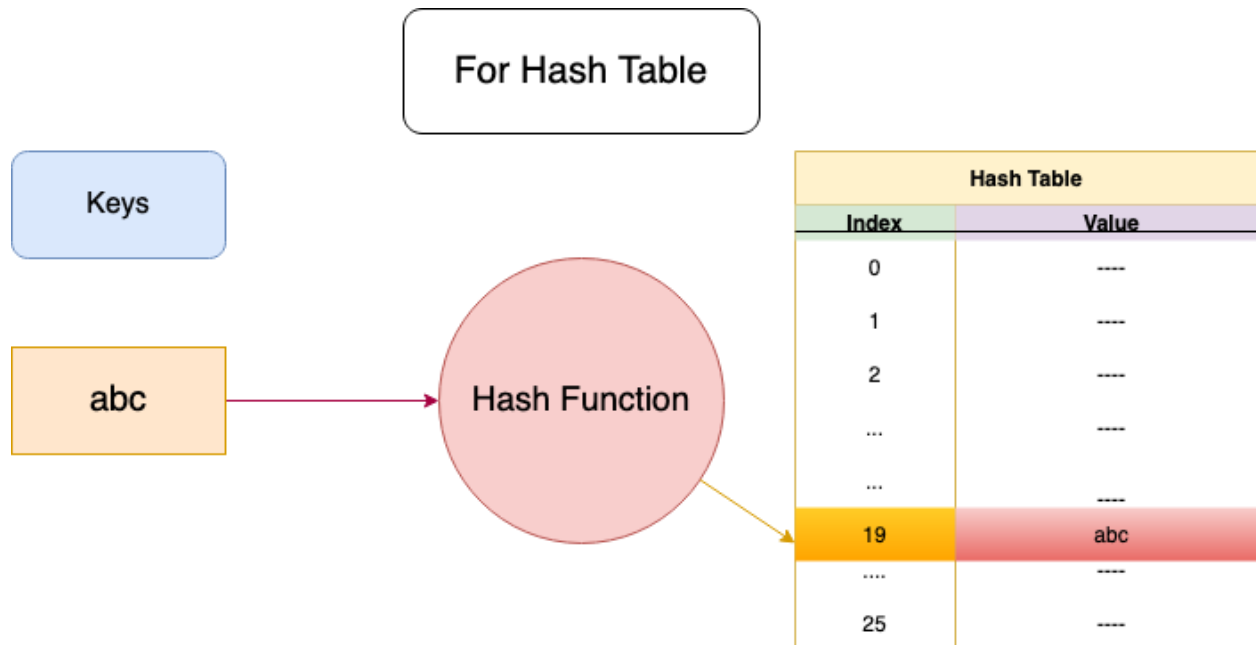
//finds the address of the given string name

```
Node* Table_Lookup(string name)
{
    int index = HashFunction(name);
    if(table[index] != NULL && name == (table[index] -> name))
    {
        return table[index];
    }
    else
    {
        return NULL;
    }
}
```

Output Screenshot:



Working structure:



⇒ **Discussion:**

- Out of the two programs, **Hashtable** is **better** as it is faster and helps in reducing the complexity.
- Advantage is that quick search is possible and the disadvantage is that hashing is complicated to implement.
- In case of linkedlist, Insertion is fast $O(1)$, but lookup is slow for large tables – $O(n)$ on average