University of Asia Pacific

CSE 430

Compiler Design Lab

Assignment – 1

Write a Program for Symbol Table

Name: Md. Azim Islam

Registration: 19201026

Section: A2

Course Code: CSE 430

Semester: Spring 2023

Baivab Das

Lecturer, University of Asia Pacific, Dhaka

CODE

```
from functools import reduce
import sys

#Symbol Table

class Node:
    #Node class
    def __init__(self, name, typ, size, dim, loc, addr):
        self.name = name
        self.typ = typ
        self.size = size
        self.dim = dim
        self.loc = loc
        self.addr = addr
        self.next = None

class LinkedList:
    def __init__(self) -> None:
        #An empty Linked List is created with a fake head node.
        self.head = Node("0Head", 0, 0, 0 ,0 ,0)

    def add(self, node):
        #We get the last node in the chain
        #Then attach the current node to the last node
        t = self.getTail(node.name)
        #If duplicate nodes with the same name exist, we throw a duplicate
error.
        if t:
            t.next = node
            node.next = None
        else:
            print("Error: A symbol already exists with the same name!\nTry
Updating the symbol.\n")

    def getTail(self, name):
        curr_node = self.head

        while curr_node.next:
            curr_node = curr_node.next
            #If duplicate nodes with the same name exist, we return None.
            if curr_node.name == name:
```

```
            return None

        return curr_node

    def search(self, name):
        #We iterate through the whole linked list to find a specific node by name.
        curr_node = self.head

        while curr_node.next:
            curr_node = curr_node.next
            if curr_node.name == name:
                return curr_node
        else:
            return None

    def update(self, name, kwargs):
        #name -> str
        #kwargs -> dict
        #we search and update a node by it's name!
        node = self.search(name)
        if node:
            #if no nodes exists in of the given name, we do not update!
            for k in kwargs:
                if k == 'name':
                    node.name = kwargs['name']
                if k == 'typ':
                    node.typ = kwargs['typ']
                if k == 'size':
                    node.size = kwargs['size']
                if k == 'loc':
                    node.loc = kwargs['loc']
                if k == 'dim':
                    node.dim = kwargs['dim']
                if k == 'addr':
                    node.addr = kwargs['addr']
        else:
            print('Error NODE not found!')

    def delete(self, name):
        print(f"Deleting: {name}")
        n1 = self.head
        n2 = n1.next
```

```
        while n2:
            if n2.name == name:
                #We cut and stitch the linked list.
                n1.next = n2.next.next if n2.next else None
                break
            else:
                n1 = n2
                n2 = n2.next

    def print(self):
        #We print all the nodes in this specific linklist chain.
        #If no node exist in a specific linked link then we print nothing.
        node = self.head.next
        while node:
            print(node.name, node.typ, node.size, node.dim, node.loc,
node.addr)
            print("Node Hash: ", Hash_Name(node.name), '\n')
            node = node.next

#A array of Linked List Inititalizing each as empty.
sym_table = [LinkedList() for i in range(1024)]


def Hash_Node(node):
    #We cumulatively multiply each of the node's name character ascii
values
    #Then MOD it by 1024 to get the index of the symbol table
    hash_value = reduce(lambda x, y : x*y, [ord(i) for i in node.name])%1024
    return hash_value


def Hash_Name(name):
    #same as above but calculates has directly for name
    #Similar to overloading
    hash_value = reduce(lambda x, y : x*y, [ord(i) for i in name])%1024
    return hash_value


#To input from a file
# for line in open('input', 'r'):
#To input from the terminal
for line in sys.stdin:
    line = line.strip()
```

```
OP = line.split(',')[0]
#name, typ, size, dim, loc, addr
if OP == 'insert':
    name, typ, size, dim, loc, addr = line.split(', ')[1: ]
    node = Node(name, typ, size, dim, loc, addr)
    print(f"Inserting: {name}")
    hzh = Hash_Node(node) #Hashed value.
    sym_table[hzh].add(node) #Inserting the node to the symbol table.


if OP == 'show':
    print("-"*50)
    print("Full Symbol Table")
    print("-"*50)
    print("name, typ, size, dim, loc, addr")
    print("-"*50)
    for i in range(1024):
        sym_table[i].print() #Printing each chain of the linked lists.
    print("-"*50)

if OP == 'update':
    name, typ, size, dim, loc, addr = line.split(', ')[1: ]
    hzh = Hash_Name(name)
    d = {'name':name, 'typ':typ, 'size':size, 'dim':dim, 'loc':loc, 'addr':addr}
    print(f"Updating: {name}")
    sym_table[hzh].update(name, d)

if OP == 'delete':
    name = line.split(', ')[1]
    hzh = Hash_Name(name)
    sym_table[hzh].delete(name)
```

INPUT

```
insert, x, ID, 2, 1, 5, 0x6dfed4
show
insert, x, ID, 2, 1, 5, 0x6dfed4
show
insert, y, ID4, 2, 3, 6, 0x6dfe23
show
update, y, ID26, 19, 20, 1, 0x6622
show
delete, y
show
```

OUTPUT

```
Inserting: x
------------------------------------------------
Full Symbol Table
------------------------------------------------
name, typ, size, dim, loc, addr
------------------------------------------------
x ID 2 1 5 0x6dfed4
Node Hash: 120

------------------------------------------------
Inserting: x
Error: A symbol already exists with the same name!
Try Updating the symbol.

------------------------------------------------
Full Symbol Table
------------------------------------------------
name, typ, size, dim, loc, addr
------------------------------------------------
x ID 2 1 5 0x6dfed4
Node Hash: 120

------------------------------------------------
Inserting: y
------------------------------------------------
Full Symbol Table
------------------------------------------------
name, typ, size, dim, loc, addr
------------------------------------------------
x ID 2 1 5 0x6dfed4
Node Hash: 120

y ID4 2 3 6 0x6dfe23
Node Hash: 121

------------------------------------------------
Updating: y
------------------------------------------------
Full Symbol Table
```

```
------------------------------------------------
        name, typ, size, dim, loc, addr
------------------------------------------------
            x ID 2 1 5 0x6dfed4
              Node Hash: 120

           y ID26 19 20 1 0x6622
              Node Hash: 121

------------------------------------------------
              Deleting: y
------------------------------------------------
              Full Symbol Table
------------------------------------------------
        name, typ, size, dim, loc, addr
------------------------------------------------
            x ID 2 1 5 0x6dfed4
              Node Hash: 120

------------------------------------------------
```