SYMBOL TABLE LAB REPORT

Name: Harsh Gupta

ROLL.NO: AP21110011001

Section: CSE-P

TITLE:

Symbol Table Implementation

STATEMENT:

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various identifiers such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler. Symbol table can be implemented in one of the following ways:

• Linear (sorted or unsorted) list

• Binary Search Tree

• Hash table

• And other ways.

In this lab session, you are required to analyse the various implementations. You need to write

code for at least two ways of implementation. Test your code with different test cases. Submit

a report of your analysis and executable code by the end of the session.

PROCEDURE:

I written the code in c programming language,first i take a character input(EXPRESSION) and i am creating two array to store the values

and the adress of that values in the pointer array and if the input is alphabet it prints identifier for this i am using is

isalpha() function and if it is not i am comparing with some operators using if block if it is a special character it will

print operator.

CODE 1:

```
//AP21110011001
```

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
int main()
{
 int x=0, n, i=0,j=0,p=0;
 void *ptr,*id_address[5];
 char ch,id_Array2[15],id_Array3[15],c;
 printf("Input the expression ending with ; sign:");
 char s[20];
 scanf("%s",s);
 while(s[i]!=';')
 {
  id_Array2[i]=s[i];
  i++;
 }
 n=i-1;
 printf("\n Symbol Table display\n");
 printf("Symbol \t addr \t\t\t type");
 while(j<=n)
 {
  c=id_Array2[j];
  if(isalpha(c))
  {
   ptr=malloc(c);
   id_address[x]=ptr;
   id_Array3[x]=c;
   printf("\n %c \t %p \t identifier\n",c,ptr);
   x++;
   j++;
  }
  else
  {
   ch=c;
   if(ch=='+'||ch=='-'||ch=='*'||ch=='/' || ch=='%'|| ch=='='|| ch=='<' || ch=='>')
   {
    ptr=malloc(ch);
    id_address[x]=ptr;
    printf("\n %c \t %p \t operator\n",ch,ptr);
    x++;
    j++;
   }
             }
}
             return 0;
}
```

INPUT:

```
        s=a+b;
```

OUTPUT:

```
        Input the expression ending with ; sign:s=a+b;

 Symbol Table display
Symbol  addr                type
 s      0x562b28f42ac0      identifier


 =      0x562b28f42b40      operator


 a      0x562b28f42b90      identifier


 +      0x562b28f42c00      operator
 b          0x562b28f42c40           identifier
```

CONCLUSION:

Symbol table is a data structure used by the compiler, where each identifier in program’s source code is stored

along with information associated with it relating to its declaration.

CODE 2:

```
//AP21110011001
class TreeNode {
    String key;
    int value;
    TreeNode left;
    TreeNode right;

    public TreeNode(String key, int value) {
        this.key = key;
        this.value = value;
        this.left = null;
        this.right = null;
    }
}

class BinarySearchTree {
    TreeNode root;

    public void insert(String key, int value) {
        root = insertRecursive(root, key, value);
    }
```

```
private TreeNode insertRecursive(TreeNode current, String key, int value) {
    if (current == null) {
        return new TreeNode(key, value);
    }
    if (key.compareTo(current.key) < 0) {
        current.left = insertRecursive(current.left, key, value);
    } else if (key.compareTo(current.key) > 0) {
        current.right = insertRecursive(current.right, key, value);
    }
    return current;
}

public Integer search(String key) {
    return searchRecursive(root, key);
}

private Integer searchRecursive(TreeNode current, String key) {
    if (current == null || current.key.equals(key)) {
        return current != null ? current.value : null;
    }
    if (key.compareTo(current.key) < 0) {
        return searchRecursive(current.left, key);
    }
    return searchRecursive(current.right, key);
}

public static void main(String[] args) {
    BinarySearchTree symbolTable = new BinarySearchTree();
    symbolTable.insert("variable1", 42);
    System.out.println(symbolTable.search("variable1")); // Output: 42
    }
}


/*
OUTPUT -42
*/
```