Department of Computer Science and Engineering

Compiler Design Lab (CS 306L)

Week 2: Symbol Table Implementation

Mula Sai Charan Reddy

AP2111010988

CSE_P

1. 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.

```
1. #include<stdio.h>
2. #include<ctype.h>
3. #include<stdlib.h>
4. int main()
5. {
6.  int x=0, n, i=0,j=0,p=0;
7.  void *ptr,*id_address[5];
8.  char ch,id_Array2[15],id_Array3[15],c;
```

```
9.    printf("Input the expression ending with ; sign:");
10.  char s[20];
11.  scanf("%s",s);
12.  while(s[i]!=';')
13.  {
14.   id_Array2[i]=s[i];
15.   i++;
16.  }
17.  n=i-1;
18.  printf("\n Symbol Table display\n");
19.  printf("Symbol \t addr \t\t\t type");
20.  while(j<=n)
21.  {
22.   c=id_Array2[j];
23.   if(isalpha(c))
24.   {
25.    ptr=malloc(c);
26.    id_address[x]=ptr;
27.    id_Array3[x]=c;
28.    printf("\n %c \t %p \t identifier\n",c,ptr);
29.    x++;
30.    j++;
31.   }
32.   else
33.   {
34.    ch=c;
35.    if(ch=='+'||ch=='-'||ch=='*'||ch=='/' || ch=='%'|| ch=='='|| ch=='<' || ch=='>')
36.    {
37.     ptr=malloc(ch);
38.     id_address[x]=ptr;
39.     printf("\n %c \t %p \t operator\n",ch,ptr);
40.     x++;
41.     j++;
42.    }
43.            }
44. }
45.            return 0;
46. }
```

```
2. 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
    }
}
```