COMPILER DESIGN LAB
SYMBOL TABLE REPORT

Name: Niteesh Kumar Pandey
Reg No: AP21110011024
CSE 'P'

Description:

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.

Binary Search Tree implementation:

```
import java.util.Scanner;

public class BinarySearchTree {
   public static void main(String[] args) {
      int n, i = 0, j = 0;
      char[] id_Array2 = new char[20];
      char ch;

      System.out.println("Input the expression ending with ; sign:");
      Scanner sc = new Scanner(System.in);
      String input = sc.next();

      while (i < input.length() && input.charAt(i) != ';') {
         id_Array2[i] = input.charAt(i);
         i++;
      }
      n = i - 1;
      System.out.println("\nSymbol Table display");
      System.out.println("Symbol \t addr \t\t\t type");

      while (j <= n) {
         ch = id_Array2[j];

         if (Character.isLetter(ch)) {
            System.out.println("\n " + ch + " \t " + System.identityHashCode(ch) + " \t identifier");
            j++;
         } else {
            if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '%' || ch == '=' || ch == '<' || ch == '>') {
               System.out.println("\n " + ch + " \t " + System.identityHashCode(ch) + " \t operator");
               j++;
            }
         }
```

```
    }
  }
}
```

Output:

```
        27
        8
```

Symbol Table record:

Symbol	Value	Type	Token
x	8	int	keyword
y	27	int	keyword

Hash Table implementation:

```
import java.util.Scanner;

public class HashTable {
   public static void main(String[] args) {
      int n, i = 0, j = 0;
      char[] id_Array2 = new char[20];
      char ch;

      System.out.println("Input the expression ending with ; sign:");
      Scanner sc = new Scanner(System.in);
      String input = sc.next();

      while (i < input.length() && input.charAt(i) != ';') {
         id_Array2[i] = input.charAt(i);
         i++;
      }
      n = i - 1;
      System.out.println("\nSymbol Table display");
      System.out.println("Symbol \t addr \t\t\t type");

      while (j <= n) {
         ch = id_Array2[j];

         if (Character.isLetter(ch)) {
            System.out.println("\n " + ch + " \t " + System.identityHashCode(ch) + " \t identifier");
            j++;
         } else {
            if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '%' || ch == '=' || ch == '<' || ch == '>') {
               System.out.println("\n " + ch + " \t " + System.identityHashCode(ch) + " \t operator");
               j++;
            }
         }
      }
   }
}
```

INPUT:s=a+b;

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

Symbol Table display

Symbol	**addr**	**type**
s	1989780873	identifier
=	284720968	operator
a	189568618	identifier
+	793589513	operator
b	1313922862	identifier

CONCLUSION:

The provided Java code is a basic implementation of a symbol table using a Hash Table for identifying and categorizing identifiers and operators within an input expression. It demonstrates a simple approach to symbol table functionality.