**Week 2 compiler design lab**

**AP20110010604**

**J. Nikhil Sai**

**This code is for creating Symbol Table using linear list.**

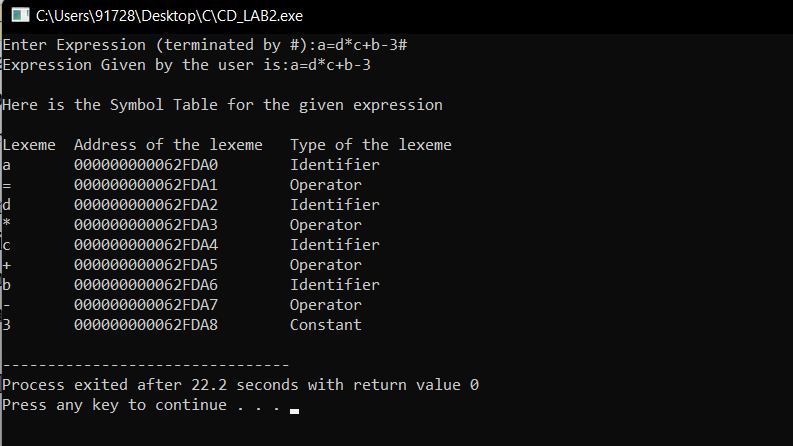
Linear list is a simplest and easiest to implement data structure. We use a single array to store names and their associated information. New names are added to the list in the order in which they are encountered. To insert a new name, we must scan down the list to make sure that it is not already there. When the name is located, the associated information can be found in words following next. Searching of names is done in order pointed by the link of the link field. A pointer “First” is maintained to point to the first record of the symbol table. Insertion is fast O(1), but lookup is slow for large tables – O(n) on average.

In these symbol table program. I am taking an expression as “a=d\*c+b-3”. Based on these in expression I am going to separate the expression into the lexemes. And based on these lexemes the code separates them and identifies which one is identifier or operator or constant or an Invalid token.

Here this program considers any alphabet as the identifier, any number from 0 to 9 as the constant and the symbols “+, -, \*, /, =” are considered as Operators.

Here this code stores the expression in expr variable which is initialized as an array of size 100.

Now the code identifies the lexemes type and stores in the Symbol Table, here is the Output of the Code.



**This code is for creating Symbol Table using Hashing.**

In hashing scheme, two tables are maintained – a hash table and symbol table and are the most commonly used method to implement symbol tables. A hash table is an array with an index range: 0 to table size – 1. These entries are pointers pointing to the names of the symbol table. To search for a name, we use a hash function that will result in an integer between 0 to table size – 1. Insertion and lookup can be made very fast – O (1). The advantage is quick to search is possible and the disadvantage is that hashing is complicated to implement.

Performance of hash table

Performance issues on using different collision resolution schemes.

Hash table size must be adequately larger than the maximum number of possible entries.

Frequently used variables should be distinct.

• Keywords or reserved words.

• Short names, e.g., i, j and k.

• Frequently used identifiers, e.g., main.

Uniformly distributed.

OUTPUT

