Lab Assignment 3

Implementation of Symbol Table Using Data Structures

a. Symbol Table Implementation Using Stack Data Structure

Language Used: Python

Steps :

1. Create a stack data structure, an empty list for storing the information of identifiers, and a tuple for storing the types of identifiers whose information can be stored in the symbol table.
2. A while loop encloses the main part of the program, User enters his/her choice whether to continue entering the identifiers or not
    a. If the user decides to continue, A menu will be displayed to show the list of identifiers accepted. The user will choose one and begin providing the information about that identifier
        i. **Variable -** The user should provide the name, data type, scope of the variable
        ii. **Function -** The user should provide the name, return type, no. of parameters followed by the name and data type of the parameters, and the parameter passing technique of the function
        iii. **Structure -** The user should provide the name, no. of members followed by the name and data type of the members of the structure
        iv. **Pointer -** The user should provide the name, data type, name of the identifier the pointer is holding
        v. **Array -** The user should provide the name, data type, no. of dimensions followed by the values of the dimensions of the array
3. In each case, the information is pushed into the stack along with the address of these identifiers
4. If the identifier with the same name is entered again, Then the output is displayed that the identifier already exists

5. After entering the information about an identifier, The list item is appended to the symbol table and the stack is cleared to push the values of the next identifier

Printing The Symbol Table :

1. Filters representing the type of identifier is placed as the first item in every list element in the symbol table
2. A for loop is used to go through all the list elements in the symbol table, if the filter is equal to ‘variable’, all the similar identifiers are printed and the loop goes through the elements to print the remaining identifiers

b. IMPLEMENTATION OF SYMBOL TABLE USING HASH TABLE:

Symbol tables are mostly implemented as hash tables, where the source code symbol itself is treated as a key for the hash function and the return value is the information about the symbol.

The following possible information about identifiers are stored in the symbol table:
The name (as a string)
Attribute: Reserved word, Variable name, Type name, Procedure name, Constant name
The data type
The block-level Its scope (global, local, or parameter)
Its offset from the base pointer (for local variables and parameters only)

A symbol table provides the following operations:
insert() and lookup()

From our code: The entry of symbol table is
{
int add;
char label[10];

}sy[11];

A compiler maintains multiple block levels of symbol tables:
Level 0: A null hash table at level 0
Level 1: Keyword in the hash table at level 1
Level 2: Global symbol table which can be accessed by all the procedures
Level 4: Scope symbol tables that are created for each scope in the program

int create(int num) is used to create the symbol table, The maximum number of values that can be given in our symbol table is 11.

int key;
key=num%11;
return key; -------> The source code to represent where given values will be stored in the table.

void search()------> is used to search the required value given.

c. Symbol Table implementation using a Linear list data structure

Language used: C

It is the easiest way to implement the symbol table

Steps :

1. We should create an array, it can be a one dimensional or multidimensional array.
2. When we enter any information, it will be stored as a name and attribute in the array which we have created.
3. At the end of an array, there can be a pointer indicating the availability to insert an element.
4. So, when we insert any input, it gets stored in the available array location, where the pointer indicates.
5. The searching is the same as linear search in an array, it starts from the first entry and goes until the last element.
6. Insertion is fast O(1), but lookup is slow for large tables – O(n) on average.