Session 3: Symbol Table Construction and Management

I. OBJECTIVES:

The main purpose of this session is to introduce the symbol table, the table in which all the identifiers are stored along with information about them. When a variable is declared, the compiler enters it as a new entry in the symbol table. When a variable is referred to in an expression, the compiler looks up in the symbol table to retrieve necessary information about it, such as its data type, value, etc., and the compiler performs other actions on the table like delete, update and so on.

II. DEMONSTRATION OF USEFUL RESOURCES:

Sample programs will be demonstrated related to the symbol table.

III. LAB EXERCISE:

Sept 1 and Step 2 of the Assignment #3 described below.

IV. ASSIGNMENT #3:

Suppose, a given C source program has been scanned, filtered and then lexically analyzed as it was done in Session 1 & 2. We have all the lexemes marked as different types of tokens like keywords, identifiers, operators, separators, parentheses, numbers, etc. We get corrected the unknown lexemes first, and then generate a Symbol Table describing the features of the identifiers. Finally, we generate a modified token stream in accordance with the Symbol Table for processing by the next phase, that is, Syntactic Analysis.

Sample source program:

```
// A program fragment
float x1 = 3.125;
/* Definition of the
function f1 */
double  f1(int x)
{
double z;
  z = 0.01+x*5.5;
return z;
}
//* Beginning of 'main'
int main(void)
{
int n1;  double z;
 n1=25;  z=f1(n1);
```

Sample input based on the program fragment:

[kw float] [id x1] [op =] [num 3.125] [sep ;] [kw double] [id f1] [par (] [kw int] [id x] [par)] [brc {] [kw double] [id z] [sep ;] [id z] [op =] [num 0.01] [op +] [id x] [op *] [num 5.5] [sep ;] [kw return] [id z] [sep ;] [brc }] [kw int] [id main] [par (] [kw void] [par)] [brc {] [kw int] [id n1] [sep ;] [kw double] [id z] [sep ;] [id n1] [op =] [num 25] [sep ;] [id z] [op =] [id f1] [par (] [id n1] [par)] [sep ;]

Sample input based on the program fragment:

[kw float] [id x1] [op =] [num 3.125] [sep ;] [kw double] [id f1] [par (] [kw int] [id x] [par)] [brc {] [kw double] [id z] [sep ;] [id z] [op =] [num 0.01] [op +] [id x] [op *] [num 5.5] [sep ;] [kw return] [id z] [sep ;] [brc }] [kw int] [id main] [par (] [kw void] [par)] [brc {] [kw int] [id n1] [sep ;] [kw double] [id z] [sep ;] [id n1] [op =] [num 25] [sep ;] [id z] [op =] [id f1] [par (] [id n1] [par)] [sep ;]

Step 1: After complete recognition of all the lexemes only identifiers are kept in pairs for formation of Symbol Tables. The token stream should look like the one as follows:

[float][id x1][=][3.125][;][double][id f1][(][int][id x][)][{][double][id z][;][id z][=][0.01][+][id x][*][5.5][;][return][id z][;][}][int][id main][(][void][)][{][int][id n1][;][double][id z][;][id n1][=][25][;][id z][=][id f1][(][id n1][)][;]

Step 2: Symbol Table generation:

Sample source program:

```
// A program fragment
float x1 = 3.125;
/* Definition of the
function f1 */
double  f1(int x)
{
double z;
  z = 0.01+x*5.5;
return z;
}
//* Beginning of 'main'
int main(void)
{
int n1;  double z;
 n1=25;  z=f1(n1);
```

Symbol Table:

1	x1	var	float	global
2	f1	func	double	global
3	x	var	int	f1
4	z	var	double	f1
5	main	func	int	global
6	n1	var	int	main
7	z	var	double	main

Step 3: Your program should implement the following functions on symbol table.

1. *insert()*
2. *update()*
3. *delete()*
4. *search()*
5. *display()*

Step 4: Modified token stream for Syntactic Analysis:

Sample source program:

```
// A program fragment
float x1 = 3.125;
/* Definition of the
function f1 */
double  f1(int x)
{
double z;
  z = 0.01+x*5.5;
return z;
}
//* Beginning of 'main'
int main(void)
{
int n1;  double z;
 n1=25;  z=f1(n1);
```

[float][id 1][=][3.125][;][double][id 2][(][int][id 3][)][{][double] [id 4][;][id 4][=][0.01][+][id 3][*][5.5] [;] [return][id 4][;][}][int] [id 5][(][void][)][{][int][id 6][;][double][id 7][;][id 6][=][25][;][id 7][=][id 2] [(][id 6][)][;]