Comipler 1

Lecture 5: Symbol table

BY: Assist Prof. Dr. Ielaf O Abdul Majjed

Symbol table

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities 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.

A symbol table may serve the following purposes depending upon the language in hand:

- To store the names of all entities in a structured form at one place.
- To verify if a variable has been declared.
- To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
- To determine the scope of a name (scope resolution).

Symoble table is used by various phases of the compiler as follows:-

1. **Lexical Analysis**: Creates new table entries in the table, for example like entries about tokens.
2. **Syntax Analysis**: Adds information regarding attribute type, scope, dimension, line of reference, use, etc in the table.
3. **Semantic Analysis**: Uses available information in the table to check for semantics i.e. to verify that expressions and assignments are semantically correct(type checking) and update it accordingly.

4. Intermediate Code generation: Refers symbol table for knowing how much and what type of run-time is allocated and table helps in adding temporary variable information.

5. Code Optimization: Uses information present in the symbol table for machine-dependent optimization.

6. Target Code generation: Generates code by using address information of identifier present in the table.

Symbol Table entries: Each entry in the symbol table is associated with attributes that support the compiler in different phases.

1- **variable name**: must always resides in the symbol table since it is the means by which a particular variable is identified for semantic analysis & code generation. This attribute should be inserted into the symbol table during lexical analysis.

2- **Object-code address (object time address)**: dictates the relative location for value(s) of a variable at run time. The address is entered in the symbol table when the variable is referenced in the source program. Insertion is performed during semantic while recalling aids in the production of correct target code for a given source program.

3- **Type**: the type of a variable is used for both – An indication of the amount of memory that must be allocated to a variable of run time (e.g. int 2 byte, char 1 byte). It is the task of the semantic when handling declaration statement – for semantic checking (e.g. it is semantically inconsistent to multiply a string by a number).

4- Number-of-dimensions & Number-of-parameters: simple variables (i.e. scalars) are considered to be of dimension 0, vectors of dimension 1, matrices of dimension 2, procedure of dimension equal to the number of parameters for that procedure.

The objective of dimension attribute is two fold:

- As a parameter in a generalized formula for calculating the address of a particular array element.
- For semantic checking (e.g., The number of dimensions in array reference should agree with the number specified in the declaration of the array). Also, the number of parameters in a procedure call must also agree with the number used in the procedure declaration.

This attribute is entered in the table when declaration of variable & procedure statements are encountered by the (semantic analysis).

5- Line declared: the source line number at which a variable is declared. (semantic task).

6- Line referenced: the source line number of all other references to the variable (semantic task).

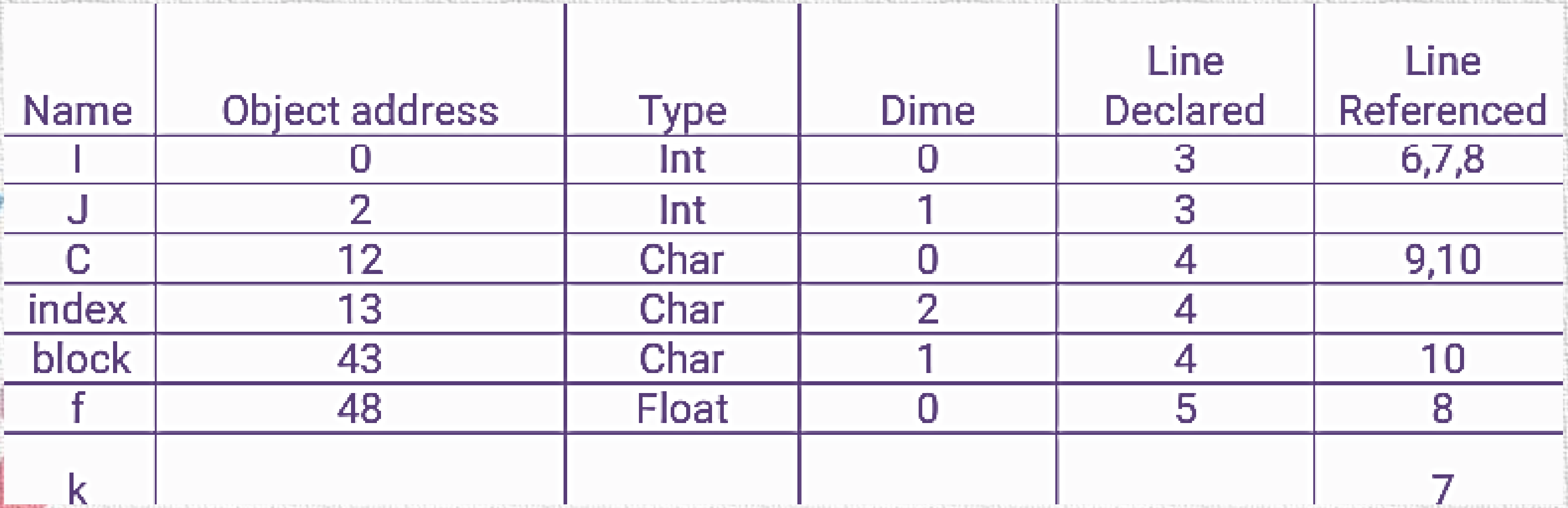
The attributes are inserted into the symbol table by performing ***insertion operation*** is required when processing a declaration statement, since a declaration is intended to be an initial description of a variables attributes in the program.

Retrieval operations are performed for all references to variables which do not involve declaration statements. The retrieved information (i.e., type, object address, dimensions, etc.) is used for semantic checking & code generation.

Retrieval operations for variable which have not been previously declared are detected at this stage & reporting appropriate error messages or warning or performing some semantic error recovery by posting a warning message & incorporating the non-declared variable in the table with as much as possible deducing the associated attributes though the context in which the variable is referenced.

EX/ Draw a diagram of a symbol table that would result when compiling the following program segment (with reporting error & warning message if any)

```
1. main();
2. {
3. int i,j[5];
4. char c, index[5][6], block[5];
5. float f;
6. i=0;
7. i=i+k;
8. f=f+i;
9. c='x';
10.block[4]=c;
11.}
```



Name	Object address	Type	Dime	Line Declared	Line Referenced
I	0	Int	0	3	6,7,8
J	2	Int	1	3	
C	12	Char	0	4	9,10
index	13	Char	2	4	
block	43	Char	1	4	10
f	48	Float	0	5	8
k					7

```
1. main();
2. {
3. int i,j[5];
4.char c, index[5][6], block[5];
5. float f;
6. i=0;
7. i=i+k;
8. f=f+i;
9. c='x';
10.block[4]=c;
11.}
```

Int: 2 byte, char: 1 byte, float: 4 byte

Erorrs and Warning

- Error line7: k is undefined (semantic)
- Warning line8: f used before initialization (semantic)
- Warning line3: j not used (semantic)
- Warning line4: index is not used (semantic)