Assignment No.11

AIM:

Generate the symbol table for language.

THEORY:

Figure 1 depicts the schematic of a two pass language processor. The first pass performs analysis of the source program and reflects its results in the intermediate representation (IR). The second pass reads and analyses the IR to perform synthesis of the target program. This avoids repeated processing of the source program.

The first pass is concerned exclusively with source language issues. Hence it is called the *front end* of the language processor. The second pass is concerned with program synthesis for a specific target language. Hence it is called the *back end* of the language processor.

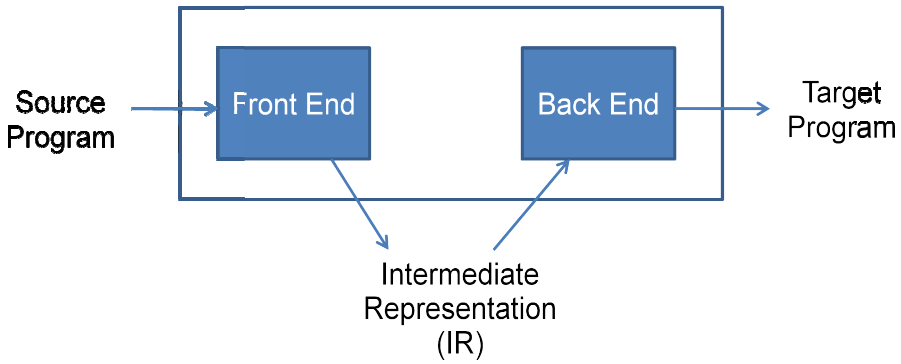

Figure 1: Two pass schematic for language processing

The Front End

The front end performs

- Lexical analysis,
- Syntax analysis and
- Semantic analysis of the source program.

Each kind of analysis involves the following functions:

1. Determine validity of a source statement from the viewpoint of the analysis.
2. Determine the 'content' of a source statement.
3. Construct a suitable representation of the source statement for use by subsequent analysis functions, or by the synthesis phase of the language processor.

The word 'content' has different connotations in lexical, syntax and semantic analysis.

- In lexical analysis, the content is the lexical class to which each lexical unit belongs,
- In syntax analysis it is the syntactic structure of a source statement.
- In semantic analysis the content is the meaning of a statement—for a declaration statement, it is the sef of attributes of a declared variable (e.g. type, length and dimensionality), while for an imperative statement, it is the sequence of actions implied by the statement.

Each analysis represents the 'content' of a source statement in the form of (1) tables of information, and (2) description of the source statement. Subsequent analysis uses this information for its own purposes and either adds information to these tables and descriptions, or constructs its own tables and descriptions.

Output of the front end

The IR produced by the front end consists of two components:

1. Tables of information
2. An *intermediate code* (IC) which is a description of the source program.

Tables

Tables contain the information obtained during different analyses of SP. The most important table is the symbol table which contains information concerning all identifiers used in the SP. The symbol table is built during lexical analysis. Semantic analysis adds information concerning symbol attributes while processing declaration statements. It may also add new names designating temporary results.

Intermediate code *(IC)*

The IC is a sequence of IC units; each IC unit representing the meaning of one action in SP. IC units may contain references to the information in various tables.

Example

Figure 2 shows the IR produced by the analysis phase for the program

```
i:  integer;
a,b: real;
a := b+i;
```

Symbol Table:

No.	Symbol	Type	Length	Address
1	i	int		
2	a	real		
3	b	real		
4	i*	real		
5	temp	real		

Intermediate code

1. Convert (id, #1) to real, giving (id, #4)
2. Add (id, #4) to (id, #3) giving (id, #5)
3. Store (id, #5) in (Id, #2)

Figure 2: Intermediate Representation

The symbol table contains information concerning the identifiers and their types. This information is determined during lexical and semantic analysis, respectively. In IC, the specification (Id, #1) refers to the id occupying the first entry in the table. i* and temp are temporary names added during semantic analysis of the assignment statement.

The Back End

The back end performs memory allocation and code generation.

Memory allocation

Memory allocation is a simple task given the presence of the symbol table. The memory requirement of an identifier is computed from its type, length and dimensionality, and memory is allocated to it. The address of the memory area is entered in the symbol table.

Example

After memory allocation, the symbol table looks as shown in Figure 3. The entries for i* and temp are not shown because memory allocation is not needed for these id's.

No.	Symbol	Type	Length	Address
1	i	int		2000
2	a	real		2001
3	b	real		2002

Figure 3: Symbol table after memory allocation

Certain decisions have to precede memory allocation, for example, whether i* and temp should be allocated memory. These decisions are taken in the preparatory steps of code generation.

Code generation

Code generation uses knowledge of the target architecture, viz. knowledge of instructions and addressing modes in the target computer, to select the appropriate instructions.

Example

For the sequence of actions for the assignment statement a :* b+i;

(i) Convert i to real, giving i*.
(ii) Add i* to b, giving temp,
(iii) Store temp in a.

the synthesis phase may decide to hold the values of i* and temp in machine registers and may generate the assembly code

```
CONV_R          AREG, I
```

```
ADD_R       AREG, B
MOVEM       AREG, A
```

where CONV_R converts the value of I into the real representation and leaves the result in AREG. ADD_R performs the addition in real mode and MOVEM puts the result into the memory area allocated to A.

PROGRAM:

INPUT:

```
#include<stdio.h>
void main()
{
        double a=10;
        int b=5;
        float c=2.3f;
        char d='A';
}
```

OUTPUT:

Generating the symbol table

ID	Datatype	Variable	Value
1	double	a	10
2	int	b	5
3	float	c	2.3f
4	char	d	'A'

CONCLUSION:

- The symbol table contains information concerning the identifiers and their types. This information is determined during lexical and semantic analysis.
- The symbol table is built during lexical analysis.

- Semantic analysis adds information concerning symbol attributes while processing declaration statements
- The symbol table is implemented in C-Language.

REFERENCES:

- Compilers - Principles, Techniques and Tools - A.V. Aho, R. Shethi and J. D. Ullman (Pearson Education)
- System Programming and operating systems – 2nd Edition D.M. Dhamdhere (TMGH)