

CS 7611 COMPILER LAB
C CODE SEMANTIC ANALYSER

TEAM MEMBERS

VAMAN VENKATESH 2017103081
A ABDUL KAREEM 2017103502

PROBLEM STATEMENT-

Writing C code is an easy task but in order to check if code written is correct , i.e, does not have redeclarations, spelling errors, or other syntax errors, the compiler performs a semantic analysis. Semantic analysis is mainly a process in compiler construction after parsing to gather necessary semantic information from the source code. Semantic analyser checks whether syntax structure constructed in the source program derives any meaning or not. Semantic analysis is also called context sensitive analysis phase. This phase performs semantic checks such as type checking (checking for type errors), or definite assignment (variable to be initialized before use), rejecting incorrect programs or issuing warnings. Our aim is to create a program using lex and yacc to identify said semantic errors and display them to the user along with the line in which they occur, we have also handled the cases of undeclared variables and variable redeclaration, additionally the semantic analyzer also checks if predefined functions (like printf, scanf, gets, getchar etc) are redefined in the program.

TOOLS USED-

- VirtualBox
- Terminal
- Gedit

IMPLEMENTATION DETAILS-

1. LEX PROGRAM

The lex file goes through each word of the input code and checks certain patterns and returns tokens to the parser. It also keeps track of the current line being read.

2. YACC PROGRAM

The yacc file has rules which have to be followed by the code, i.e, the structure or syntax of the code. In case a certain rule is not followed, the program prints the error along with the line number. If no errors occur, parsing completion message is printed. At the end of parsing, the display() function in the symbol function is called to show the identifiers.

3. SYMBOL TABLE

The semantic analyser is built by adding subroutines and C functions for the grammar rules defined in the syntax analyzer phase of the compiler. The symbol table of the syntax phase is updated in the semantic phase. At the end of checking, a symbol table is printed containing identifiers along with its scope, value and type.

OUTPUT-

1. No errors

```
#include<stdio.h>
```

```
int inc(int a)
```

```
{
```

```
    a=a+1;
```

```
    return a;
```

```
}
```

```
void main()
```

```
{
```

```
    int a,b,c;
```

```
    int b;
```

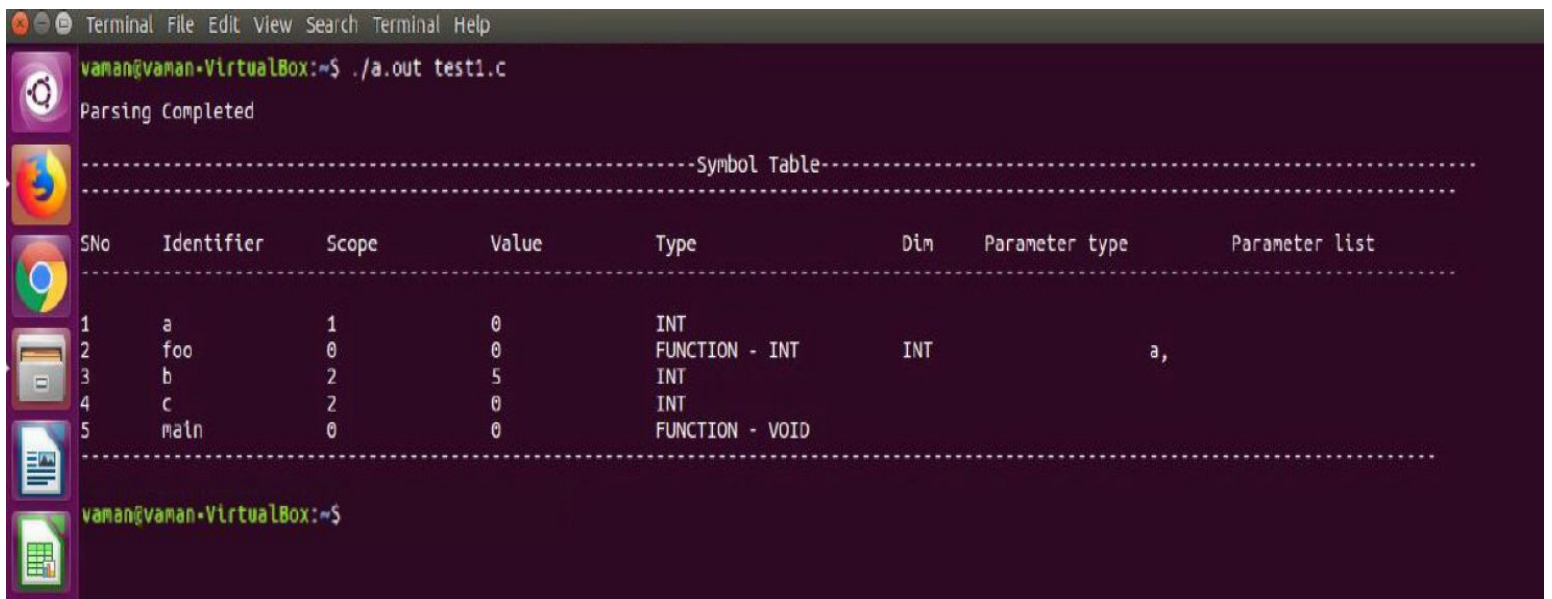
```
    b=5;
```

```
    int c;
```

```
    c=inc(b);
```

```
    return;
```

```
}
```



The screenshot shows a terminal window with a menu bar (Terminal, File, Edit, View, Search, Terminal, Help) and a sidebar with application icons. The terminal output is as follows:

```
vaman@vaman-VirtualBox:~$ ./a.out test1.c
Parsing Completed
```

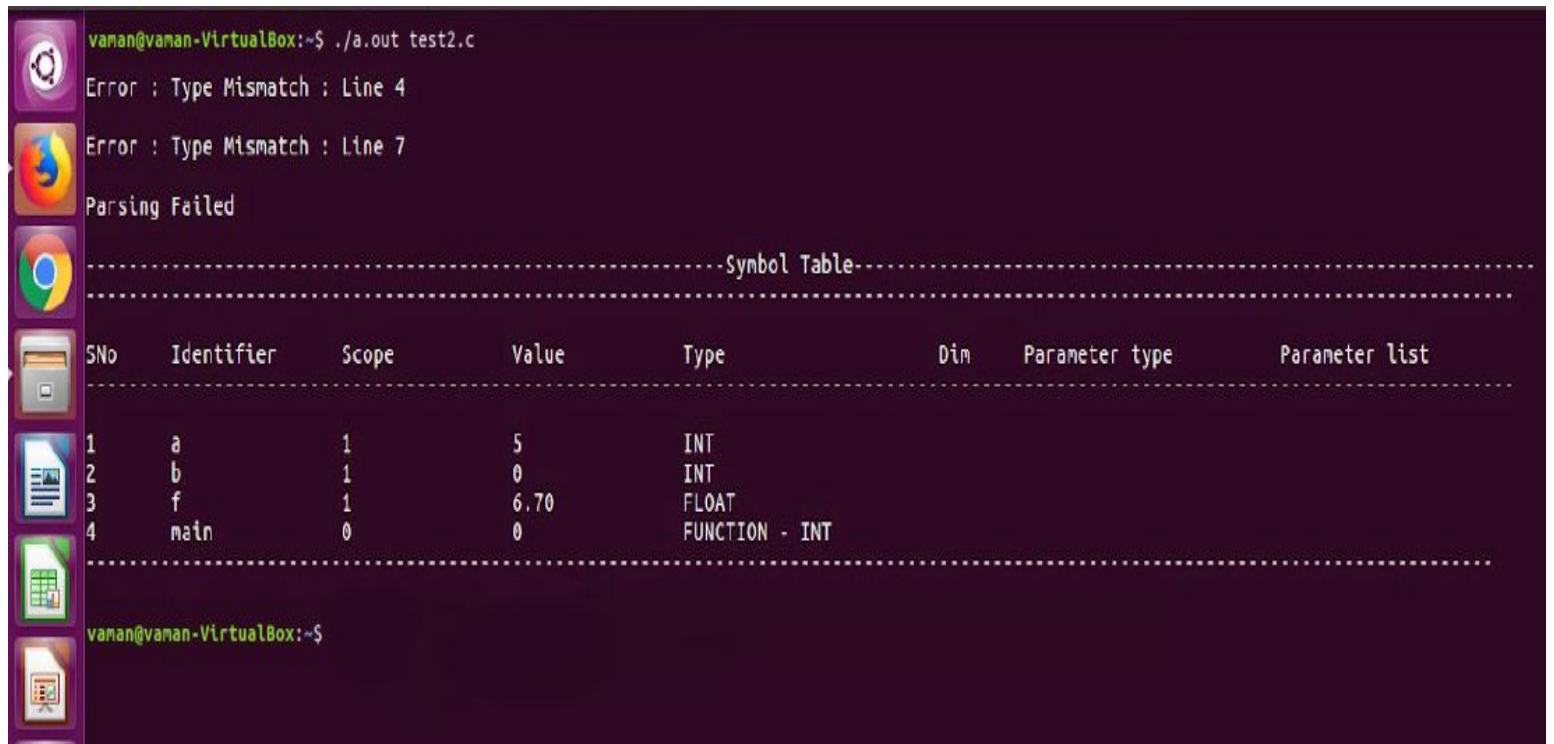
Below the output, a symbol table is displayed, enclosed in a dashed border:

SNo	Identifier	Scope	Value	Type	Dim	Parameter type	Parameter list
1	a	1	0	INT			
2	foo	0	0	FUNCTION - INT	INT		a,
3	b	2	5	INT			
4	c	2	0	INT			
5	main	0	0	FUNCTION - VOID			

The terminal prompt is now `vaman@vaman-VirtualBox:~$`.

2. Code with errors

```
#include <stdio.h>
int main()
{
int a = 5.4; //type mismatch
int b=67;
float f=6.7;
b=f; //type mismatch (int = float)
return 0;
}
```



The screenshot shows a terminal window with the following output:

```
vaman@vaman-VirtualBox:~$ ./a.out test2.c
Error : Type Mismatch : Line 4
Error : Type Mismatch : Line 7
Parsing Failed
```

Below the error messages is a symbol table:

```
-----Symbol Table-----
```

SNo	Identifier	Scope	Value	Type	Din	Parameter type	Parameter list
1	a	1	5	INT			
2	b	1	0	INT			
3	f	1	6.70	FLOAT			
4	main	0	0	FUNCTION - INT			

The terminal prompt is now:

```
vaman@vaman-VirtualBox:~$
```

GITHUB LINK-

<https://github.com/TheVams/Compiler-Project>