

# **COMPILER DESIGN LAB**

## **PROJECT – 1**

### **LEXICAL ANALYSER**

- Tejas Rafaliya – 15CO250

- Siddhartha Chowdhuri – 15CO246

# **Table of Contents**

1. **Compiler**
2. **Lexical Analysis**
3. **Syntax Analysis**
4. **Semantic Analysis**
5. **Intermediate Code Generation**
6. **Code Optimization**
7. **Code Generation**
8. **Symbol Table**
9. **Test cases for data types**
10. **Test cases for for-loops and if-conditions**
11. **Test cases for functions and function calls**
12. **Test cases for comments**

## **COMPILER:**

A compiler is computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language). Compilers are a type of translator that supports digital devices, primarily computers. The name *compiler* is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program.

The compilation process is a sequence of various phases. Each phase takes input from its previous stage, has its own representation of source program, and feeds its output to the next phase of the compiler. Let us understand the phases of a compiler.

### **Lexical Analysis:**

The first phase of scanner works as a text scanner. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. Lexical analyser represents these lexemes in the form of tokens as: <token name, attribute value>.

Sequences of characters in a token are called lexemes.

Functions of lexical analyser:

1. Tokenization i.e. dividing the program into valid tokens.
2. Remove white space characters.
3. Remove comments.
4. It also provides help in generating error message by providing row number and column number.

The lexical analyser identifies the error with the help of automation machine and the grammar of the given language on which it is based like C,C++.

### **Syntax Analysis:**

The next phase is called the syntax analysis or parsing. It takes the token produced by lexical analysis as input and generates a parse tree (or syntax tree). In this phase, token arrangements are checked against the source code grammar, i.e. the parser checks if the expression made by the tokens is syntactically correct.

### **Semantic Analysis:**

Semantic analysis checks whether the parse tree constructed follows the rules of language. For example, assignment of values is between compatible data types, and adding string to an integer. Also, the semantic analyser keeps track of identifiers, their types and expressions; whether identifiers are declared before use or not etc. The semantic analyser produces an annotated syntax tree as an output.

### **Intermediate Code Generation:**

After semantic analysis the compiler generates an intermediate code of the source code for the target machine. It represents a program for some abstract machine. It is in between the high-level language and the machine language. This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code.

### **Code Optimization:**

The next phase does code optimization of the intermediate code. Optimization can be assumed as something that removes unnecessary code lines, and arranges the sequence of statements in order to speed up the program execution without wasting resources (CPU, memory).

## **Code Generation:**

In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates the intermediate code into a sequence of (generally) re-locatable machine code. Sequence of instructions of machine code performs the task as the intermediate code would do.

## **Symbol Table:**

It is a data-structure maintained throughout all the phases of a compiler. All the identifier's names along with their types are stored here. The symbol table makes it easier for the compiler to quickly search the identifier record and retrieve it. The symbol table is also used for scope management.

## Lex code for the lexical analyser:

```
%
{
    #include <stdio.h>
    #include <string.h>
    struct symboltable
    {
        char token[100];
        char type[100];
    }table[10000];
    int i=0;
}%
letter [a-zA-Z]
digit[0-9]
%%
{digit}+("E"("+""-")?{digit}+)?
{strcpy(table[i].token,yytext);strcpy(table[i++].type,"Real Number");}
{digit}+."{digit}+("E"("+""-")?{digit}+)?
{strcpy(table[i].token,yytext);strcpy(table[i++].type,"Floating point number");}
#include <{letter}*".h>"|include<{letter}*".h>"
{strcpy(table[i].token,yytext);strcpy(table[i++].type,"Include Statement");}
"auto"|"break"|"case"|"char"|"const"|"continue"|"default"|"do"|"double"|"else"|"enum"|"
extern"|"float"|"for"|"goto"|"if"|"int"|"long"|"register"|"return"|"short"|"signed"|"
sizeof"|"static"|"struct"|"switch"|"typedef"|"union"|"unsigned"|"void"|"volatile"|"whi
le" {strcpy(table[i].token,yytext);strcpy(table[i].type,"Keyword");}
({letter}|"_")({letter}|{digit})*
{strcpy(table[i].token,yytext);strcpy(table[i++].type,"Identifer");}
"&&"|"<"|">"|"<="|">="|"="|"+"|"-|"?"|"*"|"/"|"%"|"&"|"||"
{strcpy(table[i].token,yytext);strcpy(table[i++].type,"Operator");}
"{"|"["|"(" {strcpy(table[i].token,yytext);strcpy(table[i++].type,"Opening bracket");}
"}"|"]"|")" {strcpy(table[i].token,yytext),strcpy(table[i++].type,"Closing bracket");}
"#"|"'"|"\"|"\"|",\" {strcpy(table[i].token,yytext),strcpy(table[i++].type,"Special
Character");}
\";\" {strcpy(table[i].token,yytext);strcpy(table[i++].type,"Delimiter");}
"%d"|"%s"|"%c"|"%f"|"%e\" {strcpy(table[i].token,yytext);strcpy(table[i++].type,"Format
specifier");}
\"\\n\" {strcpy(table[i].token,yytext);strcpy(table[i++].type,"New line");}
\"/\"|\"/*\"|\"*/\" {strcpy(table[i].token,yytext);strcpy(table[i++].type,"Comments");}
%%
```

```

int yywrap()
{
return 1;
}
int main()
{
yyin=fopen("abc.txt","r");
yylex();
int j;
for(j=0;j<i;j++)
{
    printf("%s - %s\n",table[j].token,table[j].type);
}
return 0;
}

```

## Screenshots of Outputs:

### 1. Test case for datatypes:

#### Input:

```

// Test case for data types, declarations and assignment statements

#include <stdio.h>

int main()
{
    char a;           //valid char assignment
    char a[100];      //valid char assignment

    int 5xyz;         //invalid identifier
    int x=10;         //valid int assignment
    x=2*q;            //invalid assignment

    float z=30.0;     //valid float assignment
    float @y=25.0;    //invalid float assignment
    float y=25.1.0;   //invalid float assignment

    z=x+y;            //valid statement
    return 0;
}

```

## Output:

```
tejas_000@Asus-pc ~
$ ./a.exe
// - Comments
Test - Identifier
data - Identifier
types - Identifier
, - Special Character
declarations - Identifier
and - Identifier
assignment - Identifier
statements - Identifier
#include <stdio.h> - Include Statement
main - Identifier
( - Opening bracket
) - Closing bracket
{ - Opening bracket
a - Identifier
; - Delimiter
// - Comments
valid - Identifier
assignment - Identifier
a - Identifier
[ - Opening bracket
100 - Real Number
] - Closing bracket
; - Delimiter
// - Comments
valid - Identifier
assignment - Identifier
5 - Real Number
xyz - Identifier
; - Delimiter
// - Comments
invalid - Identifier
identifier - Identifier
x - Identifier
= - Operator
10 - Real Number
; - Delimiter
// - Comments
valid - Identifier
assignment - Identifier
x - Identifier
= - Operator
2 - Real Number
* - Operator
q - Identifier
; - Delimiter
// - Comments
invalid - Identifier
assignment - Identifier
z - Identifier
= - Operator
30.0 - Floating point number
; - Delimiter
// - Comments
valid - Identifier
```

```
assignment - Identifier
@ - Lex error Invalid token
y - Identifier
= - Operator
25.0 - Floating point number
; - Delimiter
// - Comments
invalid - Identifier
assignment - Identifier
y - Identifier
= - Operator
25.1 - Floating point number
. - Special Character
0 - Real Number
; - Delimiter
// - Comments
invalid - Identifier
assignment - Identifier
z - Identifier
= - Operator
x - Identifier
+ - Operator
y - Identifier
; - Delimiter
// - Comments
valid - Identifier
statement - Identifier
0 - Real Number
; - Delimiter
} - Closing bracket
```



## 2. Test case for loops and if-conditions:

### Input:

// Test case for for loop, while loop, nested while loop and if-else statements

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

```
int main()
```

```
{
```

```
    int i,j=0;
```

```
        for (i=0;i<10;i++)                // valid for loop
```

```
    {
```

```
        printf("%d \n",i);
```

```
    }
```

```
        for [i=0;i<10;i++]                // invalid for loop
```

```
    {
```

```
        printf("%d \n",i);
```

```
    }
```

```
        while (i<20)                    // valid nested while loops
```

```
    {
```

```
        j=0;
```

```
        while (j<3)
```

```
        {
```

```
            j++;
```

```
        }
```

```
    }
```

```
        while {x<9}                    // invalid while statement
```

```
            x++;
```

```
        if (i<20)                    // valid if-else statements
```

```
    {
```

```
        printf("Hello World \n");
```

```
    }
```

```
    else if (i>20)
```

```
    {
```

```
        printf("Bye World \n");
```

```
    }
```

```
    else
```

```
        printf("Equality \n");
```

```
        if [i<20]                    // invalid if-else statements
```

```
            printf("Nothing happens \n");
```

```
}
```

## Output:

```
tejas_0008@ ASUS-PC ~  
$ ./a.exe  
// - Comments  
Test - Identifier  
loop - Identifier  
; - Special Character  
loop - Identifier  
; - Special Character  
nested - Identifier  
loop - Identifier  
and - Identifier  
- - Operator  
statements - Identifier  
#include <stdio.h> - Include Statement  
main - Identifier  
( - Opening bracket  
) - Closing bracket  
{ - Opening bracket  
i - Identifier  
; - Special Character  
j - Identifier  
= - Operator  
0 - Real Number  
; - Delimiter  
( - Opening bracket  
i - Identifier  
= - Operator  
0 - Real Number  
; - Delimiter  
i - Identifier  
< - Operator  
10 - Real Number  
; - Delimiter  
i - Identifier  
+ - Operator  
+ - Operator  
) - Closing bracket  
// - Comments  
valid - Identifier  
loop - Identifier  
{ - Opening bracket  
printf - Identifier  
( - Opening bracket  
" - Special Character  
&d - Format specifier  
\n - New line  
" - Special Character  
" - Special Character  
i - Identifier  
) - Closing bracket  
; - Delimiter  
} - Closing bracket  
{ - Opening bracket  
i - Identifier  
= - Operator  
0 - Real Number  
; - Delimiter  
i - Identifier  
- - Operator  
< - Operator  
10 - Real Number  
; - Delimiter  
i - Identifier  
+ - Operator  
+ - Operator  
] - Closing bracket  
// - Comments  
invalid - Identifier  
loop - Identifier  
{ - Opening bracket  
printf - Identifier  
( - Opening bracket  
" - Special Character  
&d - Format specifier  
\n - New line  
" - Special Character  
" - Special Character  
i - Identifier  
) - Closing bracket  
; - Delimiter  
} - Closing bracket  
( - Opening bracket  
i - Identifier  
< - Operator  
20 - Real Number  
) - Closing bracket  
// - Comments  
valid - Identifier  
nested - Identifier  
loops - Identifier  
{ - Opening bracket  
j - Identifier  
= - Operator  
0 - Real Number  
; - Delimiter  
( - Opening bracket  
j - Identifier  
< - Operator  
3 - Real Number  
) - Closing bracket  
{ - Opening bracket  
j - Identifier  
+ - Operator  
+ - Operator  
; - Delimiter  
i - Identifier  
} - Closing bracket  
{ - Opening bracket  
x - Identifier  
< - Operator  
9 - Real Number  
} - Closing bracket  
// - Comments  
invalid - Identifier  
statement - Identifier  
x - Identifier
```

```
+ - Operator
+ - Operator
; - Delimiter
( - Opening bracket
{ - Identifier
< - Operator
20 - Real Number
) - Closing bracket
// - Comments
valid - Identifier
- - Operator
statements - Identifier
{ - Opening bracket
printf - Identifier
( - Opening bracket
" - Special Character
Hello - Identifier
world - Identifier
\n - New Line
" - Special Character
) - Closing bracket
; - Delimiter
} - Closing bracket
( - Opening bracket
i - Identifier
> - Operator
20 - Real Number
) - Closing bracket
( - Opening bracket
printf - Identifier
( - Opening bracket
" - Special Character
Bye - Identifier
world - Identifier
\n - New Line
" - Special Character
) - Closing bracket
; - Delimiter
} - Closing bracket
printf - Identifier
( - Opening bracket
" - Special Character
Equality - Identifier
\n - New Line
" - Special Character
) - Closing bracket
; - Delimiter
[ - Opening bracket
i - Identifier
< - Operator
20 - Real Number
] - Closing bracket
// - Comments
invalid - Identifier
- - Operator
statements - Identifier
printf - Identifier
( - Opening bracket
```

```
" - Special Character
Nothing - Identifier
happens - Identifier
\n - New Line
" - Special Character
) - Closing bracket
; - Delimiter
} - Closing bracket
```



## Output:

```
raj@raj-ubuntu:~$ ./a.exe
// - Comments
test - Identifier
defining - Identifier
user - Identifier
- - Operator
defined - Identifier
functions - Identifier
and - Identifier
invoking - Identifier
them - Identifier
in - Identifier
the - Identifier
main - Identifier
function - Identifier
#include <stdio.h> - Include Statement
printpattern - Identifier
{ - Opening bracket
} - Closing bracket
// - Comments
correct - Identifier
function - Identifier
declaration - Identifier
{ - Opening bracket
printf - Identifier
{ - Opening bracket
" - Special Character
" - Operator
\n - New Line
" - Special Character
} - Closing bracket
} - Closing bracket
ret10 - Identifier
{ - Opening bracket
} - Closing bracket
// - Comments
incorrect - Identifier
function - Identifier
declaration - Identifier
{ - Opening bracket
10 - Real Number
; - Delimiter
} - Closing bracket
checkgl0 - Identifier
{ - Opening bracket
x - Identifier
} - Closing bracket
// - Comments
correct - Identifier
function - Identifier
declaration - Identifier
{ - Opening bracket
{ - Opening bracket
{ - Opening bracket
x - Identifier
" - Operator
10 - Real Number
```

```
10 - Real Number
} - Closing bracket
} - Closing bracket
printf - Identifier
{ - Opening bracket
" - Special Character
sd - Format specifier
is - Identifier
greater - Identifier
than - Identifier
10 - Real Number
\n - New Line
" - Special Character
" - Special Character
x - Identifier
} - Closing bracket
} - Delimiter
} - Closing bracket
printf - Identifier
{ - Opening bracket
" - Special Character
sd - Format specifier
is - Identifier
not - Identifier
greater - Identifier
than - Identifier
10 - Real Number
\n - New Line
" - Special Character
" - Special Character
x - Identifier
} - Closing bracket
; - Delimiter
} - Closing bracket
main - Identifier
{ - Opening bracket
} - Closing bracket
{ - Opening bracket
i - Identifier
; - Special Character
} - Identifier
; - Delimiter
; - Delimiter
printpattern - Identifier
{ - Opening bracket
} - Closing bracket
; - Delimiter
// - Comments
correct - Identifier
function - Identifier
call - Identifier
i - Identifier
= - Operator
ret10 - Identifier
{ - Opening bracket
} - Closing bracket
} - Delimiter
// - Comments
incorrect - Identifier
```

```
incorrect - Identifier
function - Identifier
call - Identifier
checkgl0 - Identifier
{ - Opening bracket
i - Identifier
} - Closing bracket
; - Delimiter
// - Comments
correct - Identifier
function - Identifier
call - Identifier
} - Closing bracket
```

#### 4. Test case for comments:

##### Input:

```
#include <stdio.h>

int main()
{

//This is a comment                                Valid comment

/*
This is also a comment                            Valid comment
*/

                                                    //Invalid comment

/*
This comment has no end
}
}
```

##### Output:

```
tejas_000@Asus-pc ~
$ ./a.exe
#include <stdio.h> - Include Statement
main - Identifier
( - Opening bracket
) - Closing bracket
{ - Opening bracket
// - Comments
This - Identifier
is - Identifier
a - Identifier
comment - Identifier
Valid - Identifier
comment - Identifier
/* - Comments
This - Identifier
is - Identifier
also - Identifier
a - Identifier
comment - Identifier
Valid - Identifier
comment - Identifier
*/ - Comments
// - Comments
Invalid - Identifier
comment - Identifier
/* - Comments
This - Identifier
comment - Identifier
has - Identifier
no - Identifier
end - Identifier
} - Closing bracket
```