COMPILER DESIGN LAB – MINI PROJECT

PROJECT REPORT

ON

PHASES OF COMPILER FOR C# LANGUAGE

TEAM MEMBERS		
NAME	**ROLL NUMBERS**	**SECTION**
SANDESH HEBBAR	02	B
SANKET SRIVASTAVA	31	B
SANCHIT KHETAWAT	32	B

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY, MAHE

ABOUT C#

C# is an object-oriented programming language from Microsoft that aims to combine the computing power of C++ with the programming ease of Visual Basic. C# is based on C++ and contains features similar to those of Java. C# is designed to work with Microsoft's '.Net' platform. Microsoft's aim is to facilitate the exchange of information and services over the Web, and to enable developers to build highly portable applications. C# simplifies programming through its use of Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP) which allow access to a programming object or method without requiring the programmer to write additional code for each step. Because programmers can build on existing code, rather than repeatedly duplicating it, C# is expected to make it faster and less expensive to get new products and services to market.

OBJECTIVES

The objective of the project is to create a Lexical Analyzer that will determine tokens present in the code of the chosen programming language that is given as input, followed by the construction of a Symbol Table that will store the identifiers and their respective attribute values, and finally implementing a bottom-up parser for the grammar that will parse a given input.

GRAMMAR

input_section --> using_directives namespace_member_declarations

using_directives--> using_namespace_directive | using_directives using_namespace_directive | ε

using_namespace_directive--> USING namespace_or_type_name SEMI_COLON

namespace_member_declarations--> namespace_member_declaration|
namespace_member_declarations namespace_member_declaration | ε

namespace_member_declaration--> namespace_declaration | class_declaration

namespace_declaration--> NAMESPACE qualified_identifier namespace_body semi_colon_opt

qualified_identifier--> identifier| qualified_identifier DOT identifier

identifier--> AVAILABLE_IDENTIFIER | AT identifier_or_keyword

identifier_or_keyword--> AVAILABLE_IDENTIFIER | KEYWORD

namespace_body--> OPCU namespace_member_declarations CLCU

class_declaration--> class_modifiers partial_opt CLASS identifier class_base class_body
semi_colon_opt

semi_colon_opt--> SEMI_COLON | ε

class_modifiers--> MODIFIERS| MODIFIERS class_modifiers | ε

class_base--> COLON class_type | COLON interface_type_list | COLON class_type COMMA interface_type_list | ε

class_type--> namespace_or_type_name | OBJECT| STRING

interface_type_list--> namespace_or_type_name | interface_type_list COMMA namespace_or_type_name

class_body--> OPCU class_member_declarations CLCU

class_member_declarations--> class_member_declaration | class_member_declarations class_member_declaration | ε

class_member_declaration--> method_declarator | class_declaration

method_declarator--> method_header method_body

method_header--> method_modifiers partial_opt return_type member_name OPBR method_parameters CLBR

method_parameters--> type_list identifier_list | method_parameters COMMA type identifier_list | ε

method_modifiers--> MODIFIERS| method_modifiers MODIFIERS | ε

partial_opt--> PARTIAL | ε

return_type--> type | VOID

member_name--> identifier

method_body--> block | SEMI_COLON

block--> OPCU statement_list CLCU

statement_list--> statement | statement statement_list | ε

statement--> declaration_statement assignment_statement embedded_statement

declaration_statement--> type_list identifier_list SEMI_COLON declaration_statement | ε

type_list--> type| type OPSQ number_opt CLSQ

number_opt--> NUMBER | ε

identifier_list--> identifier | identifier COMMA identifier_list

assignment_statement--> identifier ASSIGN expression SEMI_COLON | ε

expression--> simple_expression exp_prime

simple_expression--> term simp_exp_prime

term--> factor term_prime

factor--> identifier| NUMBER

term_prime--> MULOP factor term_prime | ε

simp_exp_prime--> ADDOP term simp_exp_prime | ε

exp_prime--> RELOP simple_expression | ε

embedded_statement--> block | selection_statement | iteration_statement | jump_statement | ε

selection_statement --> if_statement | switch_statement

if_statement--> IF OPBR expression CLBR embedded_statement else_block

else_block--> ELSE if_statement| ELSE embedded_statement | ε

switch_statement--> SWITCH OPBR expression CLBR switch_block

switch_block--> OPCU switch_sections_opt CLCU

switch_sections_opt--> switch_sections | ε

switch_sections--> switch_section | switch_section switch_sections

switch_section--> switch_label statement_list

switch_label--> CASE expression COLON | DEFAULT COLON

iteration_statement--> while_statement | for_statement

while_statement--> WHILE OPBR expression CLBR embedded_statement

for_statement--> FOR OPBR assignment_statement SEMI_COLON expression SEMI_COLON assignment_statement CLBR embedded_statement

jump_statement--> break_statement| continue_statement | return_statement

break_statement--> BREAK SEMI_COLON

continue_statement--> CONTINUE SEMI_COLON

return_statement--> RETURN expression SEMI_COLON

namespace_or_type_name--> identifier | namespace_or_type_name DOT identifier

type--> simple_type

simple_type--> numeric_type | BOOL

numeric_type--> INTEGRAL_TYPE | FLOATING_POINT_TYPE

LANGUAGE USED

- Flex is used to implement the lexical analyzer
    - Flex – Fast Lexical Analyzer Generator – is a computer program used for generating lexical analyzers. Therefore, it produces tokens from the given input file and will store the identifers and attribute values for each identifer into a symbol table.
- Bison is used to implement the bottom-up parser
    - Bison is a general-purpose parser generator that converts a grammar description for a context-free grammar into a C program to parse that grammar. The Bison parser is a bottom-up parser and it tries, by shifting and reducting, to reduce the entire input down to the grammar's start symbol.
- C to implement the Symbol Table
    - The symbol table is a data structure containg a record for each identifier with fields for the attributes of the identifier.

PARSER USED

The type of parser we have used is a Bottom-Up parser. Bottom-up parsers, which are also known as shift-reduce parsers, build the parse tree from leaves to the root. The parser traces a right-most derivation in reverse by starting with the input string and working backwards to the start symbol.

USER DOCUMENTATION

The lexical analyzer, which is created by the Flex Program, takes an input code that is written in C# programming language. The flex code will generate tokens that will be present in the source program (input file). When an identifier in the source program is detected by the lexical analyzer, the identifier is entered into the symbol table along with attribute values. The tokens generated from the Flex code will be passed to the Bison program. The Bison program, which will contain the context-free grammar of the language and the tokens generated by the lexical analyzer, will parse the given source program.

In order to execute the program, the user only needs to run a bash script with the required input file as command line arguement. For Ex: if sum_digits.cs is the file name, then the command would be ‘bash cSharpExec.sh sum_digits.cs’.

CODE

```
//cSharpParser.y

%{
	#include<stdio.h>
	#include<stdlib.h>
	#include<string.h>
	int yylex();
	int yyerror();
	extern FILE *yyin;
	FILE *fparse;
%}
%token USING SEMI_COLON DOT AT AVAILABLE_IDENTIFIER KEYWORD CLASS MODIFIERS COLON OBJECT STRING OPCU CLCU CONST COMMA ASSIGN QUESTION_MARK LOGICAL_OR LOGICAL_AND OR XOR AND RELOP ADDOP MULOP SHORTHAND OPBR CLBR PARTIAL VOID VAR IF ELSE SWITCH CASE DEFAULT WHILE FOR BREAK CONTINUE RETURN BOOL INTEGRAL_TYPE FLOATING_POINT_TYPE NUMBER SIZEOF INCREMENT DECREMENT STRING_LITERAL VERBATIM_STRING_LITERAL CHARACTER_LITERAL NOT NEGATE NAMESPACE OPSQ CLSQ

%define parse.error verbose

%%
input_section
: using_directives namespace_member_declarations
;

using_directives
: using_namespace_directive
| using_directives using_namespace_directive
|
;

using_namespace_directive
: USING namespace_or_type_name SEMI_COLON
;

namespace_member_declarations
: namespace_member_declaration
| namespace_member_declarations namespace_member_declaration
|
;

namespace_member_declaration
: namespace_declaration
| class_declaration
;

namespace_declaration
: NAMESPACE qualified_identifier namespace_body semi_colon_opt
;
```

```
qualified_identifier
: identifier
|  qualified_identifier  DOT  identifier
;

identifier
:  AVAILABLE_IDENTIFIER
| AT  identifier_or_keyword
;

identifier_or_keyword
:  AVAILABLE_IDENTIFIER
|  KEYWORD
;

namespace_body
:  OPCU  namespace_member_declarations  CLCU
;

class_declaration
:  class_modifiers   partial_opt CLASS  identifier  class_base  class_body  semi_colon_opt
;

semi_colon_opt
:  SEMI_COLON
|
;

class_modifiers
:  MODIFIERS
|  MODIFIERS class_modifiers
|
;

class_base
:  COLON  class_type
|  COLON  interface_type_list
|  COLON  class_type   COMMA  interface_type_list
|
;

class_type
:  namespace_or_type_name
|  OBJECT
|  STRING
;

interface_type_list
:  namespace_or_type_name
|  interface_type_list  COMMA  namespace_or_type_name
;
```

```
class_body
: OPCU class_member_declarations CLCU
;

class_member_declarations
: class_member_declaration
| class_member_declarations class_member_declaration
|
;

class_member_declaration
: method_declarator
| class_declaration
;

method_declarator
: method_header method_body
;

method_header
: method_modifiers partial_opt return_type member_name OPBR method_parameters CLBR
;

method_parameters
: type_list identifier_list
| method_parameters COMMA type identifier_list
|
;

method_modifiers
: MODIFIERS
| method_modifiers MODIFIERS
|
;

partial_opt
: PARTIAL
|
;

return_type
: type
| VOID
;

member_name
: identifier
;

method_body
: block
```

```
| SEMI_COLON
;

block
: OPCU statement_list CLCU
;


statement_list
: statement
| statement statement_list
|
;

statement
: declaration_statement assignment_statement embedded_statement
;

declaration_statement
: type_list identifier_list SEMI_COLON declaration_statement
|
;

type_list
: type
| type OPSQ number_opt CLSQ
;

number_opt
: NUMBER
|
;

identifier_list
: identifier
| identifier COMMA identifier_list
;

assignment_statement
: identifier ASSIGN expression SEMI_COLON
|
;

expression
: simple_expression exp_prime
;

simple_expression
: term simp_exp_prime
;

term
```

```
: factor term_prime
;

factor
: identifier
| NUMBER
;

term_prime
: MULOP factor term_prime
|
;

simp_exp_prime
: ADDOP term simp_exp_prime
|
;

exp_prime
: RELOP simple_expression
|
;

embedded_statement
: block
| selection_statement
| iteration_statement
| jump_statement
|
;

selection_statement
: if_statement
| switch_statement
;

if_statement
: IF OPBR expression CLBR embedded_statement else_block
;

else_block
: ELSE if_statement
| ELSE embedded_statement
|
;

switch_statement
: SWITCH OPBR expression CLBR switch_block
;

switch_block
: OPCU switch_sections_opt CLCU
```

```
;

switch_sections_opt
: switch_sections
|
;

switch_sections
: switch_section
| switch_section switch_sections
;

switch_section
: switch_label statement_list
;

switch_label
: CASE expression COLON
| DEFAULT COLON
;

iteration_statement
: while_statement
| for_statement
;

while_statement
: WHILE OPBR expression CLBR embedded_statement
;

for_statement
: FOR OPBR assignment_statement SEMI_COLON expression SEMI_COLON
assignment_statement CLBR embedded_statement
;

jump_statement
: break_statement
| continue_statement
| return_statement
;

break_statement
: BREAK SEMI_COLON
;

continue_statement
: CONTINUE SEMI_COLON
;

return_statement
: RETURN expression SEMI_COLON
;
```

```
namespace_or_type_name
:  identifier
|  namespace_or_type_name   DOT  identifier
;

type
:  simple_type
;

simple_type
:  numeric_type
|  BOOL
;

numeric_type
:  INTEGRAL_TYPE
|  FLOATING_POINT_TYPE
;

%%


int yyerror(char *msg){
	printf("\n\n*******\nError: %s\n*******\n", msg);
	return 1;
}


void main(int argc, char* argv[]){

	char str[100] = "../Input_Files/";
	strcat(str, argv[1]);
	printf("%s\n", str);

	yyin = fopen(str, "r");

	do{
		if(yyparse()){
			printf("\nParsing Terminated.... Exiting\n\n");
			exit(0);
		}
	}while(!feof(yyin));

	printf("\nInput has been parsed successfully...\n");
	printf("\nTokens generated from the input are show in 'Token_Generation/tokens.txt'\n");
}
```

```
//cSharpParser.l

%{
	#include "cSharpParser.tab.h"
	#include "../Token_Generation/token_gen.h"
	int row =1, col = 1;
%}
%%
"using" {printf("%s", yytext); initialize_tokens();  generateTokens(yytext, row, col); col+=strlen(yytext); return USING;}

";" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return SEMI_COLON;}

"." {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return DOT;}

"@" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return AT;}

"class" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CLASS;}

("static"|"new"|"public"|"protected"|"internal"|"private"|"abstract") {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return MODIFIERS;}

":" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return COLON;}

"object" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return OBJECT;}

"{" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return OPCU;}

"}" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CLCU;}

"[" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return OPSQ;}

"]" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CLSQ;}

"const" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CONST;}

"," {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return COMMA;}

"=" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return ASSIGN;}

"?" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return QUESTION_MARK;}

"||" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return LOGICAL_OR;}

"&&" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return LOGICAL_AND;}

"|" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return OR;}

"^" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return XOR;}

"&" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return AND;}

("=="|"!="|"<"|">"|"<="|">=") {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return RELOP;}

("+"|"-") {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return ADDOP;}

("*"|"/"|"%") {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return MULOP;}

("+="|"-="|"*="|"/="|"%=") {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return SHORTHAND;}

"(" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return OPBR;}
```

")" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CLBR;}

"partial" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return PARTIAL;}

"void" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return VOID;}

"var" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return VAR;}

"if" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return IF;}

"else" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return ELSE;}

"switch" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return SWITCH;}

"case" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CASE;}

"default" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return DEFAULT;}

"while" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return WHILE;}

"for" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return FOR;}

"break" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return BREAK;}

"continue" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CONTINUE;}

"return" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return RETURN;}

("sbyte"|"byte"|"int"|"char"|"string") {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return INTEGRAL_TYPE;}

("float"|"double") {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return FLOATING_POINT_TYPE;}

"bool" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return BOOL;}

[0-9]+ {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return NUMBER;}

"sizeof" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return SIZEOF;}

"namespace" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return NAMESPACE;}

"!" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return NOT;}

"~" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return NEGATE;}

"++" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return INCREMENT;}

"--" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return DECREMENT;}

"\'"[a-zA-Z]"\'" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return CHARACTER_LITERAL;}

"\""(.)*"\"" {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return STRING_LITERAL;}

[a-zA-Z_][a-zA-Z0-9_]* {printf("%s", yytext); generateTokens(yytext, row, col); col+=strlen(yytext); return AVAILABLE_IDENTIFIER;}

. {printf("%s", yytext); col++;}

"\n" {printf("%s", yytext); row++; col = 1;}

%%

int yywrap()

{ return 1; }

INPUT SNAPSHOT

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5
6   namespace Program
7   {
8       public class Program
9       {
10          static void Main(string[] args)
11          {
12              int num, sum, r;
13              num = 231;
14              sum = 0;
15              double sq;
16              while (num != 0)
17              {
18                  r = num % 10;
19                  num = num / 10;
20                  sum = sum + r;
21              }
22          }
23
24          int square(int num){
25              return num*num;
26          }
27      }
28  }
```

Fig 1: Input File : sum_digits.cs

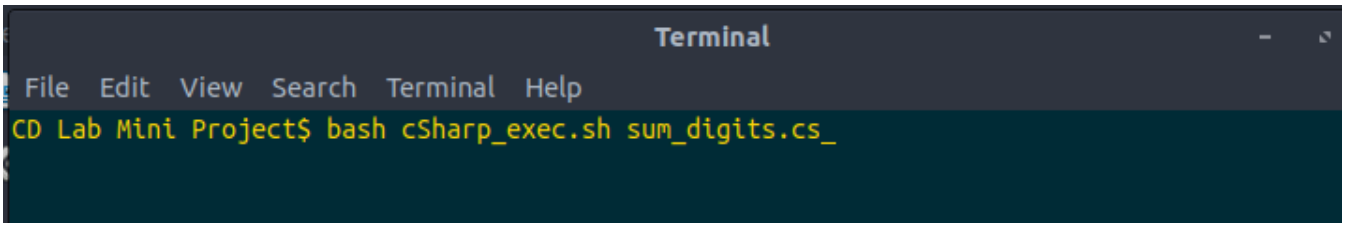

Fig 2: Terminal script execution command

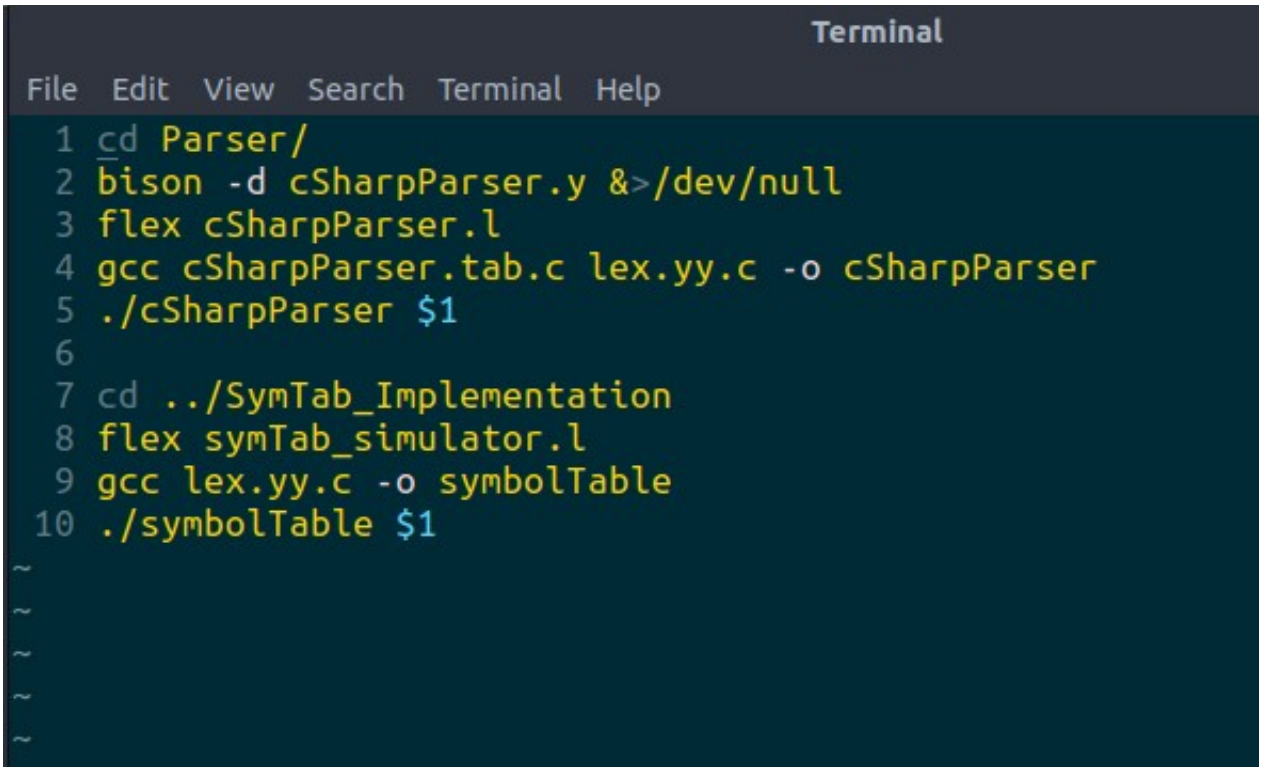

Fig 3: Shell script

OUTPUT SNAPSHOT

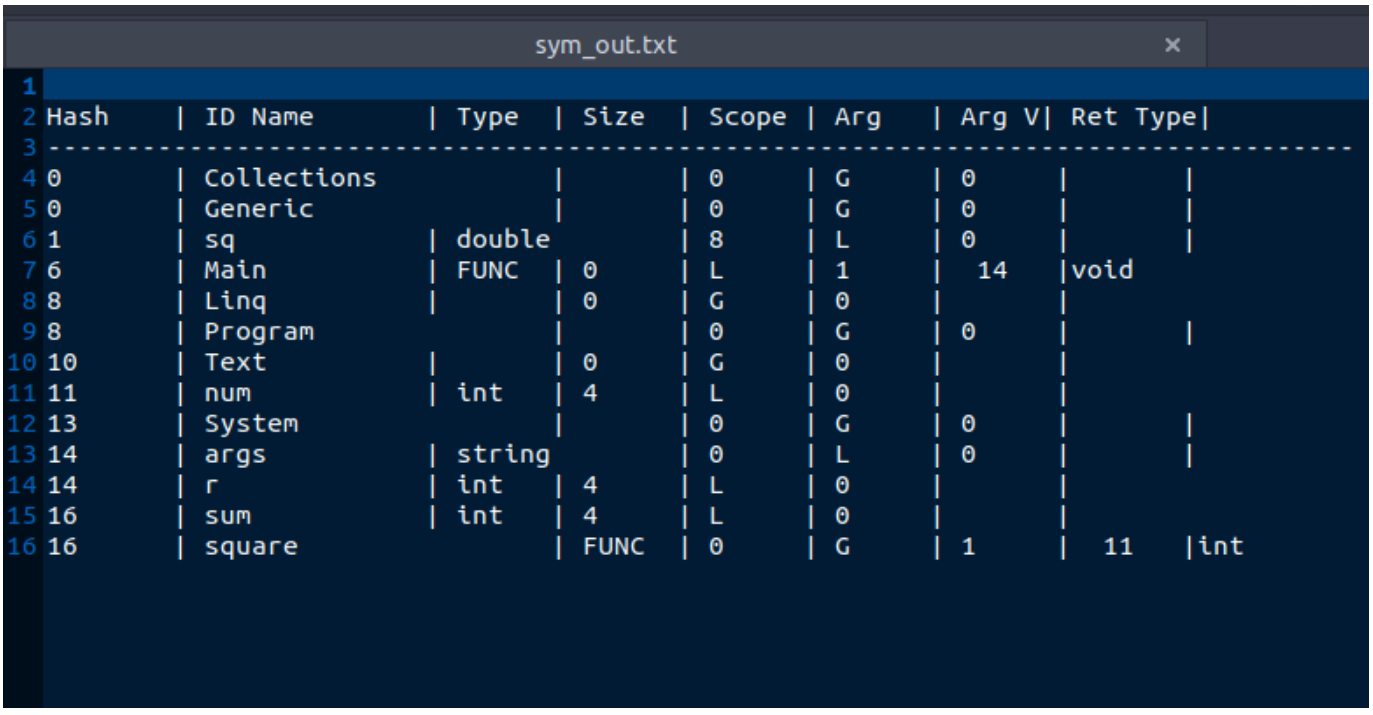

Fig 4: Generated Symbol Table

```
Terminal
File  Edit  View  Search  Terminal  Help
CD Lab Mini Project$ bash cSharp_exec.sh sum_digits.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Program
{
    public class Program
    {
        static void Main(string[] args)
        {
            int num, sum, r;
            num = 231;
            sum = 0;
            double sq;
            while (num != 0)
            {
                r = num % 10;
                num = num / 10;
                sum = sum + r;
            }
        }

        int square(int num){
            return num*num;
        }
    }
}
Input has been parsed successfully...

Tokens generated from the input are show in 'Token_Generation/tokens.txt'

Symbol Table has been generated(File : SymTab_Implementation/sym_out.txt)
CD Lab Mini Project$ _
```

Fig 5: Output generated on the terminal

Fig 6: Tokens generated using FLEX

```
<using , 1 , 1>
<System , 1 , 7>
<; , 1 , 13>
<using , 2 , 1>
<System , 2 , 7>
<. , 2 , 13>
<Collections , 2 , 14>
<. , 2 , 25>
<Generic , 2 , 26>
<; , 2 , 33>
<using , 3 , 1>
<System , 3 , 7>
<. , 3 , 13>
<Linq , 3 , 14>
<; , 3 , 18>
<using , 4 , 1>
<System , 4 , 7>
<. , 4 , 13>
<Text , 4 , 14>
<; , 4 , 18>
<namespace , 6 , 1>
<Program , 6 , 11>
<{ , 7 , 1>
<public , 8 , 5>
<class , 8 , 12>
<Program , 8 , 18>
<{ , 9 , 5>
<static , 10 , 9>
<void , 10 , 16>
<Main , 10 , 21>
<( , 10 , 25>
<string , 10 , 26>
<[ , 10 , 32>
<] , 10 , 33>
<args , 10 , 35>
<) , 10 , 39>
<{ , 11 , 9>
<int , 12 , 13>
<num , 12 , 17>
<, , 12 , 20>
<sum , 12 , 22>
<, , 12 , 25>
<r , 12 , 27>
<; , 12 , 28>
<num , 13 , 13>
<= , 13 , 17>
<231 , 13 , 19>
<; , 13 , 22>
<sum , 14 , 13>
<= , 14 , 17>
<0 , 14 , 19>
<; , 14 , 20>
<double , 15 , 13>
<sq , 15 , 20>
<; , 15 , 22>
<while , 16 , 13>
<( , 16 , 19>
<num , 16 , 20>
<!= , 16 , 24>
<0 , 16 , 27>
<) , 16 , 28>
<{ , 17 , 13>
<r , 18 , 17>
<= , 18 , 19>
<num , 18 , 21>
<% , 18 , 25>
<10 , 18 , 27>
<; , 18 , 29>
<num , 19 , 17>
<= , 19 , 21>
<num , 19 , 23>
```