NATIONAL INSTITUTE OF TECHNOLOGY, DELHI

COMPILER DESIGN PROJECT
MINI-C COMPILER
(CSC)

WORKED ON, BY : VADLAMUDDI NEELVITTAL BHARATH (191210053)
VINAY CHOUDHARY (191210059)
VINAY JAISWAL (191210060)

UNDER GUIDANCE OF : DR. SHELLY SACHDEVA
NEHA BANSAL
KANIKA SONI
PRABHAT PUSHP

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

INDEX

ABSTRACT

OBJECTIVE

The main goal of this project is to design a mini compiler for a subset of the C language as part of the Compiler Design Lab(CO351) course. The compiler is to be built in four phases finishing at the Intermediate Code Generation Phase. The subset of the C language chosen is to include certain data types, constructs and functions as mentioned in the specifications below. The implementation will be carried out with LEX and YACC.

PHASES OF THE PROJECT

- Implementation of Scanner/Lexical Analyser
- Implementation of Parser
- Implementation of Semantic Checker for C language
- Intermediate Code generation for C language

MINI-COMPILER SPECIFICATIONS :

The compiler is going to support the following cases :

1. **Keywords** - int, break, continue, else, for, void, goto, char, do, if, return, while.

2. **Identifiers** - identified by the regular expression (_ |{letter})({letter}|{digit}| _){0,31}.

3. **Comments** - single line comments (specified with // or /* ... */), multi-line comments (specified with /* ... */).
4. **Strings** - can identify strings mentioned in double quotes.

5. **Preprocessor directives** - can identify filenames (stdio.h) after #include.

6. **Data types** - int,char (supports comma declaration).

7. **Arrays** - int A[n]
   - Syntax
     - int A[3]={1,2,3};

8. **Punctuators** - [ ] , <> , {} , , , : , = , ; , # , ” ” , ’ ’.

9. **Operators** - arithmetic (+, - , * , /) ,increment(++) and decrement(--), assignment (

=).

10. **Condition** - if else

- Syntax:
  - if (condition == true){

    //code

    }

    else{

    //code

    }

11. **Loops -**

    - Syntax

- while(condition){

  //code
  }

- for(initialization;condition;increment/decrement){
  //code
  }

- while(condition){
  //code
  }

- do{
  //code

  }while(condition);

  The loop control structures that are supported are break,continue and goto.

12. **Functions-**

Void functions with no return type and a single parameter will be implemented

○ Syntax

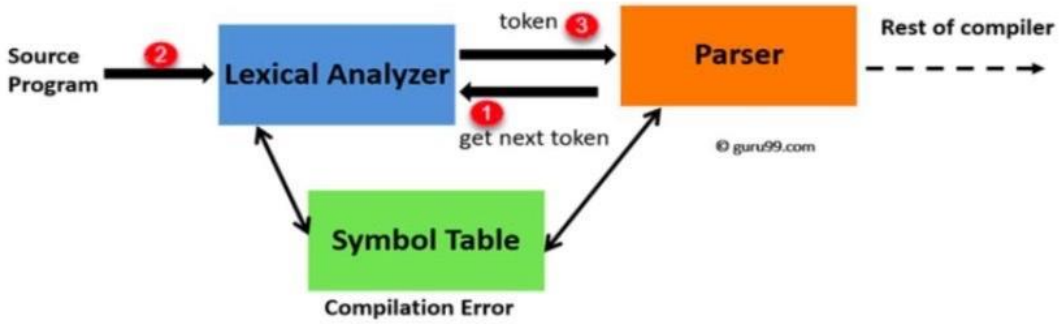
■ void sample_function(int a){

//code

}

The mini compiler will be implemented in a straightforward fashion , using Lex and YACC tools starting off with the Scanner as the first module. If time permits we will add more features to cover a larger subset of the C language.

ARCHITECTURE AND WORKING :



TOOLS TO BE USED :

YACC, LEX

DOMAIN OF THE COMPILER :

The mini-compiler handles most of the cases of the C Language :

1. Identifies and removes comments.
2. Identifies the various operators in the language.
3. Checks for the validity of the identifiers.
4. Identifies the types of the variables, strings and numeric constants.
5. Ignore white spaces.
6. Able to identify the keywords, function definitions and loops.

Syntax is handled by yacc where grammar rules are specified for the entire language.

Semantics are handled using semantic rules for type checking while performing operations to ensure operations are valid.

SAMPLE INPUT AND OUTPUT :

⇨ Input 1

```c
/* The lexical analyser should remove all the valid comments and throw an exception for all the invalid comments */

#include<stdio.h>

int main(){

    // Single line comment

    /* Single line comment with different beginning */

    /* Multi-line comment

    should be removed by lex */

    /* Nested comments are /* not */ allowed */

  return 0;

}

```

⇨ Output 1

```
D:\Mini C\Phase 1>output test-case-comments.c
        <stdio.h>                                       : Preprocessor directive
        int                                             : Keyword
        main                                            : Keyword
        (                                               : Open Round Bracket
        )                                               : Closed Round Bracket
        {                                               : Open Curly Bracket
Error 14: Nested comments are invalid
Lexical analysis finished
```

⇨ Input 2

```c
#include<stdio.h>
//Illegal header format
#include

int main(){
  //Illegal identifier
  int 2invalid;
  //Identifier too long
  int abcdefghijklmnopqrstuvwxyzabcdefgh;
  int x , y;
  //Illegeal character
  y = x $ x;
  /* Nested comments /*are not*/ allowed */
  return 0;
}
```

⇨ Output 2

```
D:\Mini C\Phase 1>output test-case-errors.c
        <stdio.h>                                                   : Preprocessor directive
Error 2: Header format not allowed
        int                                                         : Keyword
        main                                                        : Keyword
        (                                                           : Open Round Bracket
        )                                                           : Closed Round Bracket
        {                                                           : Open Curly Bracket
        int                                                         : Keyword
Error 6: Illegal identifier format
        ;                                                           : Delimiter
        int                                                         : Keyword
Error 8: Identifier too long,must be between 1 to 32 characters
        ;                                                           : Delimiter
        int                                                         : Keyword
        x                                                           : Identifier
        ,                                                           : Comma
        y                                                           : Identifier
        ;                                                           : Delimiter
        y                                                           : Identifier
        =                                                           : Assignment Operator
        x                                                           : Identifier
Error 11: Illegal character
        x                                                           : Identifier
        ;                                                           : Delimiter
Error 12: Nested comments are invalid
Lexical analysis finished
```

⇨ Input 3

```c
#include<stdio.h>
#include "string.h"

int main(){

    printf("This is a valid string.");
    printf("This string does not terminate);
    return 0;
}

```

⇨ Output 3

```
D:\Mini C\Phase 1>output test-case-strings.c
        <stdio.h>                                       : Preprocessor directive
        "string.h"                                      : Preprocessor directive
        int                                             : Keyword
        main                                            : Keyword
        (                                               : Open Round Bracket
        )                                               : Closed Round Bracket
        {                                               : Open Curly Bracket
        printf                                          : Identifier
        (                                               : Open Round Bracket
        "This is a valid string."                       : String
        )                                               : Closed Round Bracket
        ;                                               : Delimiter
        printf                                          : Identifier
        (                                               : Open Round Bracket
Error 6: Illegally terminated string
Lexical analysis finished
```

⇨ Input 4

```c
/* This file tests the detection of allowed keywords , identifiers and other tokens such as punctuators , operators */

//Identifies prepocessor directives
#include<stdio.h>
#include "stdlib.h"
//Identifies macro preprocessor directives
#define MAX 100

int main(){

    //Identifies keywords int , long , char , if , else and operators
    int a;
    long int b;
    char c;
    int e,f,g;

    //Identifies constants
    int f = 0xAB;
    c = 'A';
    f = 1;
    g = -5;

    //Identifies arithmetic operators
    e = f+g;
    e = f-g;

    // Identifies conditions
    if(c=='A')
        a = 10;
    else
        a = 30;

    // Identifies loops

    for ( a = 0; a<3; a++){
    }

    return 0;
}

```

⇨ Output 4

```
D:\Mini C\Phase 1>output test-case-tokens.c
        <stdio.h>                                       : Preprocessor directive
        "stdlib.h"                                      : Preprocessor directive
        MAX                                             : Macropreprocessor directive
        100                                             : Integer Constant
        int                                             : Keyword
        main                                            : Keyword
        (                                               : Open Round Bracket
        )                                               : Closed Round Bracket
        {                                               : Open Curly Bracket
        int                                             : Keyword
        a                                               : Identifier
        ;                                               : Delimiter
        long                                            : Keyword
        int                                             : Keyword
        b                                               : Identifier
        ;                                               : Delimiter
        char                                            : Keyword
        c                                               : Identifier
        ;                                               : Delimiter
        int                                             : Keyword
        e                                               : Identifier
        ,                                               : Comma
        f                                               : Identifier
        ,                                               : Comma
        g                                               : Identifier
        ;                                               : Delimiter
        int                                             : Keyword
        f                                               : Identifier
        =                                               : Assignment Operator
        0xAB                                            : Hexadecimal Constant
        ;                                               : Delimiter
        c                                               : Identifier
        =                                               : Assignment Operator
        'A'                                             : Character
        ;                                               : Delimiter
        f                                               : Identifier
        =                                               : Assignment Operator
        1                                               : Integer Constant
        ;                                               : Delimiter
        g                                               : Identifier
        =                                               : Assignment Operator
```

```
-5                                                  : Integer Constant
;                                                   : Delimiter
e                                                   : Identifier
=                                                   : Assignment Operator
f                                                   : Identifier
+                                                   : Arithmetic Operator
g                                                   : Identifier
;                                                   : Delimiter
e                                                   : Identifier
=                                                   : Assignment Operator
f                                                   : Identifier
-                                                   : Arithmetic Operator
g                                                   : Identifier
;                                                   : Delimiter
if                                                  : Keyword
(                                                   : Open Round Bracket
c                                                   : Identifier
==                                                  : Comparison Operator
'A'                                                 : Character
)                                                   : Closed Round Bracket
a                                                   : Identifier
=                                                   : Assignment Operator
10                                                  : Integer Constant
;                                                   : Delimiter
else                                                : Keyword
a                                                   : Identifier
=                                                   : Assignment Operator
30                                                  : Integer Constant
;                                                   : Delimiter
for                                                 : Keyword
(                                                   : Open Round Bracket
a                                                   : Identifier
=                                                   : Assignment Operator
0                                                   : Integer Constant
;                                                   : Delimiter
a                                                   : Identifier
<                                                   : Comparison Operator
3                                                   : Integer Constant
;                                                   : Delimiter
a                                                   : Identifier
++                                                  : Increment Operator
)                                                   : Closed Round Bracket
{                                                   : Open Curly Bracket
}                                                   : Closed Curly Bracket
```

```
        0                                                       : Integer Constant
        ;                                                       : Delimiter
        }                                                       : Closed Curly Bracket

Printing Symbol Table


---------------------------------------

        (lexeme, token)
        (    e, IDENTIFIER)
        (    f, IDENTIFIER)
        (    g, IDENTIFIER)
        (   if, KEYWORD)
        ( char, KEYWORD)
        ( main, KEYWORD)
        ( else, KEYWORD)
        (  for, KEYWORD)
        (  int, KEYWORD)
        ( long, KEYWORD)
        (return, KEYWORD)
        (    a, IDENTIFIER)
        (    b, IDENTIFIER)
        (    c, IDENTIFIER)
---------------------------------------
```

```
Printing Constant Table


-----------------------------------------

        (lexeme, token)
        (  100, INT_CONSTANT)
        (    0, INT_CONSTANT)
        (    1, INT_CONSTANT)
        (    3, INT_CONSTANT)
        (   10, INT_CONSTANT)
        (   -5, INT_CONSTANT)
        (   30, INT_CONSTANT)
        ( 0xAB, HEX_CONSTANT)
-----------------------------------------
Lexical analysis finished
```

REFERENCES :

1. Aho A.V, Sethi R, and Ullman J.D. Compilers: Principles, Techniques, and Tools. Addison-Wesley, 1986

2. http://www.di.univr.it/documenti/OccorrenzaIns/matdid/matdid065185.pdf

3. http://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf

4. GeeksforGeeks.

NOTE OF THANKS

We are extremely grateful to our faculty-in-charge Dr.Shelly Ma’am for providing us such a great opportunity to work on this project and learn the basics of Compiler Design.

We extend our gratitude to TA’s in-charge, Neha Basal Ma’am, Kanika Soni Ma’am and Prabhat Pushp Sir for their constant support and guidance throughout the making of the project and making it a success.