PROJECT REPORT

Submitted for the course: Theory of Computation and Compiler Design (CSE 2002)

TOPIC: DESIGNING A MINI-COMPILER FOR C-LANGUAGE

By

Name of students	Registeration No.
Sakshi Aggarwal	16BCE0254
Corrina Marieanne Barnabas	16BCE0754
Saunak Sahoo	16BCE2167
Anurag B.G.	16BCE2298
Aparna Bimal	16BCE2203

Slot:G1+TG1

Name of faculty: Debi Prasanna Acharjaya

(SCHOOL OF COMPUTER SCIENCE AND ENGINEERING)

1st November, 2017

CERTIFICATE

This is to certify that the project work entitled “Basic social networking site” that is being

Submitted by” Sakshi Aggarwal(16BCE0254) and Mahak Gupta(16BCI0012)” For

Theory of Computation and Compiler Design (CSE 2002) is a record of bonafide work done under my supervision. The contents of this project work, in full or in parts, have neither been taken from any other Source nor have been submitted for any other CAL course.

Place: VIT University, Vellore

Date: 1st November, 2017

Signature of Students: Sakshi Aggarwal (16BCE0254)
Corrrina Marieanne Barnabas(16BCE0754)
Saunak Sahoo(16BCE2167)
Anurag B.G.(16BCE2298)
Aparna Bimal(16BCE2203)

Signature of Faculty: DEBIPRASANNA ACHARJAYA

ACKNOWLEDGEMENTS

We thank our Dean, for the well-crafted education pattern due to which undertaking of this project was possible. Next, we thank our Prof, DebiPrasanna Acharjaya for always encouraging us to perform better and giving us the knowledge of the subject that played an important role in completion of project.

We also thank our classmates and friends who helped us debug some of the problems. We are grateful to our parents too for sending us to such a good Institution like VIT.

ABSTRACT

Compiler: A program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer.

In this project we will attempt to create a simple mini-version of a C compiler. The compiler that we create would check an input program for the following rules:

1. Data types: Integers, strings and characters
2. Identifiers:
   - An Identifier can only have alphanumeric characters(a-z , A-Z , 0-9) and underscore(_).
   - The first character of an identifier can only contain alphabet(a-z , A-Z) or underscore (_).
   - Not case sensitive.
   - Keywords are not allowed to be used as Identifiers.
   - No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.
   - The maximum length of an Identifier is 6 characters.
3. Arithmetic expressions:
   a. Arithmetic operators (+, -, *, /, %)
   b. Uniray operator
   c. Paranthesis
   d. Relational expressions (>, <, >=, <=, ==, !=)
4. Statements:
   a. Declarations
   b. Initialisations
   c. If-else without nesting
   d. Switch statements without nesting
   e. While/for statements
5. Input/output statements

The compiler would read the input program and based on the above specifications it would check for errors. For this we would require error handling library, resources libraries, the main compiler code and an input program. If the program that is input is correct according to the rules mentioned above, then the compiler will generate machine code and display it on the screen.

INTRODUCTION

The aim of the project is to design a basic compiler

Compilers are important an essential programming tool they improve software productivity by hiding low-level details. There are many Domain-specific languages for encapsulating domain expertise so they work as a tool for designing and evaluating computer architectures. They have inspired RISC, VLIW machines. Compilers are very important component as a machines' performance measured on compiled code. There are many techniques for developing other programming tools for Example: error detection tools – Program translation can be used to solve other problems. Binary translation (processor change, adding virtualization support) – Implementation of domain-specific languages. There modern day applications include: CAD, database, graphics, networking, bio-computation, IoT.

METHODOLOGY

What qualities are important in a compiler?
1. Correct code
2. Output runs fast
3. Compiler runs fast
4. Compile time proportional to program size
5. Support for separate compilation
6. Good diagnostics for syntax errors
7. Works well with the debugger
8. Good diagnostics for flow anomalies
9. Cross language calls
10. Consistent, predictable optimization

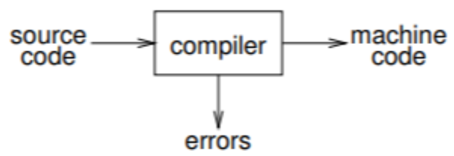


Fig1: An abstract view of working of compiler

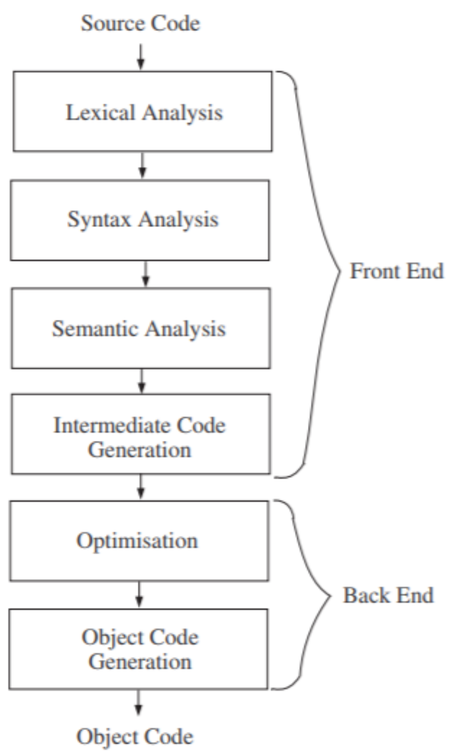
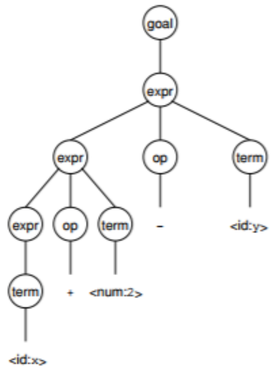


Fig2: Compilation Phases

Phase1: Lexical Analysis
Input: stream of characters from source program
Output: stream of tokens (groups of logically related characters e.g. identifiers, reserved words, punctuation, numbers, etc.). In the case of tokens such as identifiers and numbers an additional quantity (the lexeme) is required to indicate which identifier or number is represented by this instance of the token.
For example:
Input: x + 2 − y
Output: id(x), +, num(2), -, id(y)

Phase2: Syntax Analysis (Parsing)
Input: stream of tokens from lexical analysis.
Output: parse tree (groups tokens to show the structure of the given sentence).
For example: Input: id(x), +, num(2), -, id(y)
Output:



Parse trees often contain a lot of unnecessary information, so compilers often use an abstract syntax tree: + - This is much more concise. Abstract syntax trees (ASTs) are often used as an IR between front end and back end.

Phase3: Semantic Analysis
This phase checks the source program for semantic errors, and gathers type information for the intermediate code generation phase.

Phase4: Intermediate Code Generation
Intermediate code is a kind of abstract machine code. Which does not rely on a particular target machine by specifying the registers or memory locations to be used for each operation. This separates compilation into a mostly language dependent front end, and a mostly machine dependent back end

For example:

```
loop: JLE x 0 end
      SUB x 1 temp
      MOV temp x
      JMP loop
end:  ...
```

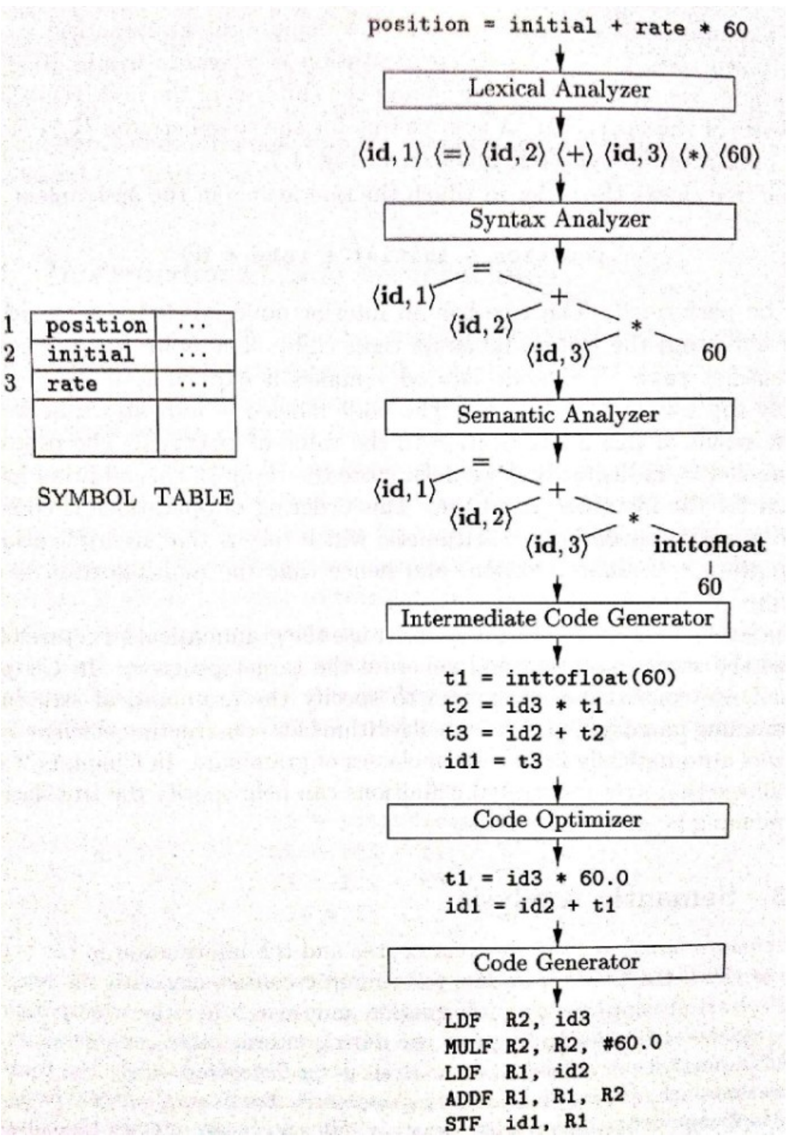
Phase5: Intermediate Code Generation

This is an optional phase which can be used to improve the improve the intermediate code to make it run faster and/or use less memory. For example, the variable temp in the previous fragment of intermediate code is not required. This can be removed to give the following:

```
loop: JLE x 0 end
      SUB x 1 x
      JMP loop
end:  ...
```

Phase6: Object Code Generation

This phase translates intermediate code into object code, allocating memory allocations for data, and selecting registers.



<u>Sample program that will add 2 numbers:</u>

```
//New.txt
declaration
int c1


start
c1=2+3
end
```

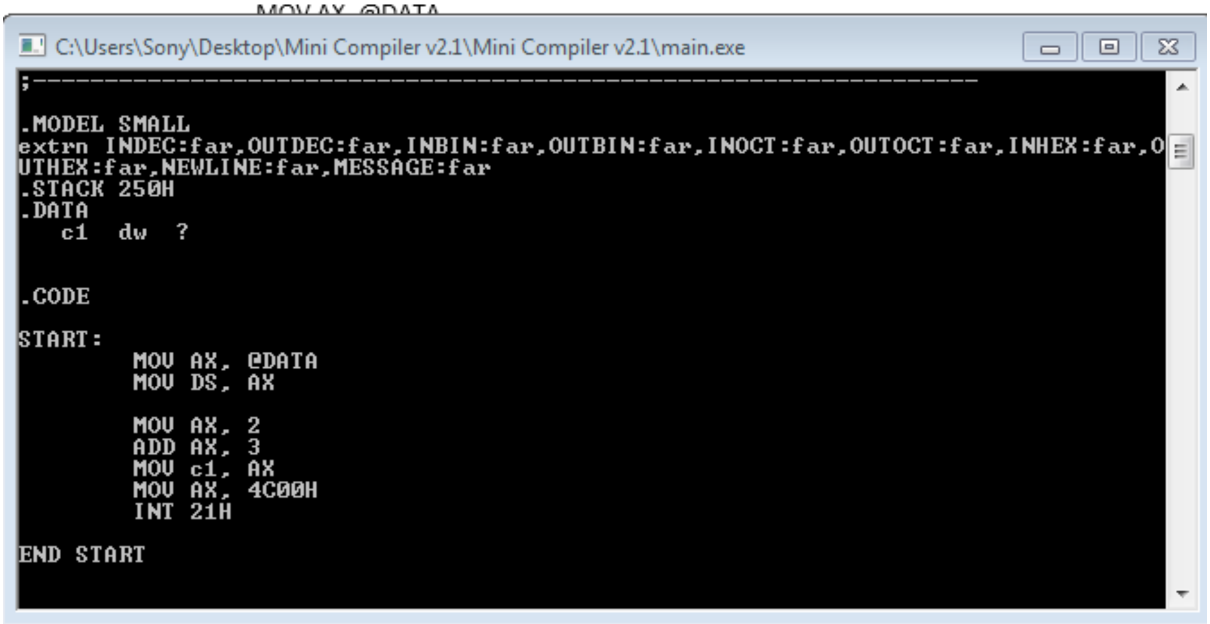
This will be pushed through the lexical analyser which generates tokens:

0.	declaration	16
1.	int	18
2.	c1	26
3.	start	23
4.	c1	26
5.	=	36
6.	2	27
7.	+	30
8.	3	27
9.	end	24

The order of tokens generated are validated and then it is confirmed that the program has been compiled successfully.

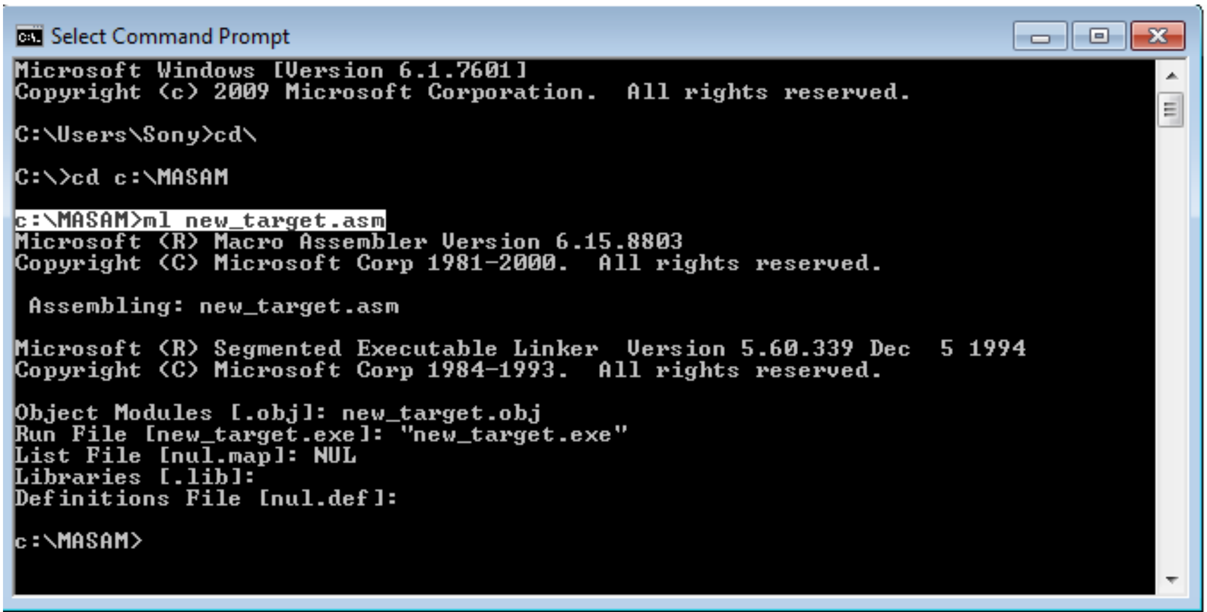
This generates an assembly language file (.asm) :

```
//new_target.asm

MODEL SMALL

extrn
INDEC:far,OUTDEC:far,INBIN:far,OUTBIN:far,INOCT:far,OUTOCT:far,INHEX:far,OUTHEX:far,NEWLINE:far,MESSAGE:far

.STACK 250H

.DATA

  c1  dw  ?



.CODE


START:

        MOV AX, @DATA

        MOV DS, AX


        MOV AX, 2

        ADD AX, 3

        MOV c1, AX

        MOV AX, 4C00H

        INT 21H


END START
```



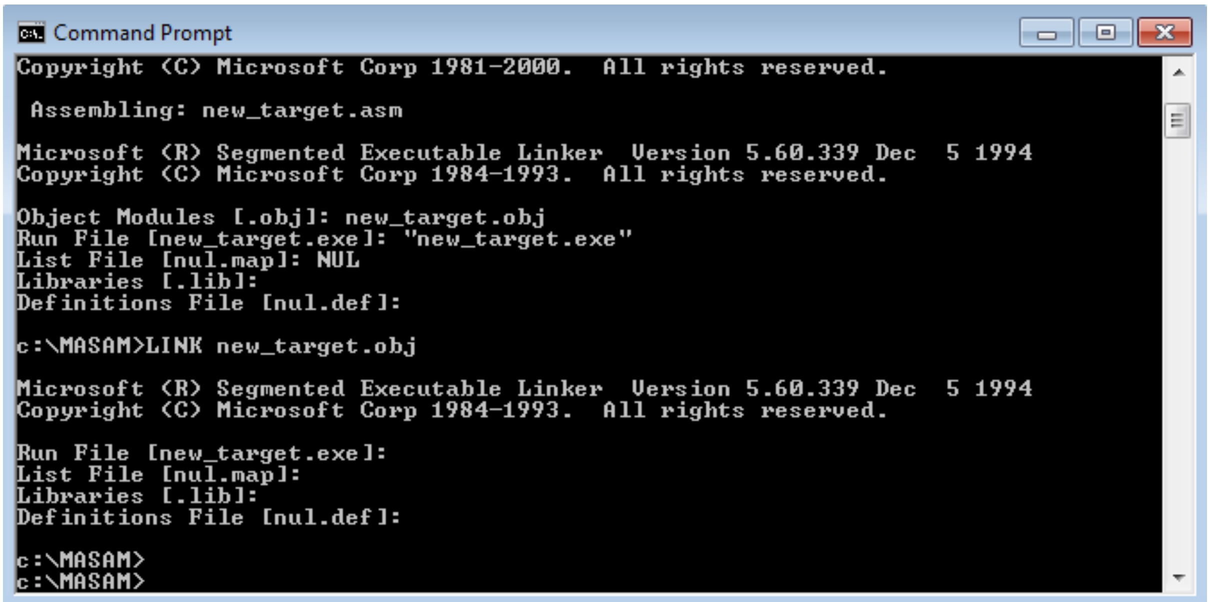
```
C:\Users\Sony\Desktop\Mini Compiler v2.1\Mini Compiler v2.1\main.exe
;-------------------------------------------------------------------

.MODEL SMALL
extrn  INDEC:far,OUTDEC:far,INBIN:far,OUTBIN:far,INOCT:far,OUTOCT:far,INHEX:far,O
UTHEX:far,NEWLINE:far,MESSAGE:far
.STACK 250H
.DATA
   c1   dw   ?


.CODE

START:
        MOV AX, @DATA
        MOV DS, AX

        MOV AX, 2
        ADD AX, 3
        MOV c1, AX
        MOV AX, 4C00H
        INT 21H

END START
```

The Assembly code is then compiled using MASAM to generate machine code (ml module).



```
Select Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sony>cd\

C:\>cd c:\MASAM

c:\MASAM>ml new_target.asm
Microsoft (R) Macro Assembler Version 6.15.8803
Copyright (C) Microsoft Corp 1981-2000.  All rights reserved.

 Assembling: new_target.asm

Microsoft (R) Segmented Executable Linker  Version 5.60.339 Dec  5 1994
Copyright (C) Microsoft Corp 1984-1993.  All rights reserved.

Object Modules [.obj]: new_target.obj
Run File [new_target.exe]: "new_target.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

c:\MASAM>
```

This generates an object file (.obj), which must be linked using LINK module with library files to get the .exe (executable) file:



```
Command Prompt
Copyright (C) Microsoft Corp 1981-2000.  All rights reserved.

 Assembling: new_target.asm

Microsoft (R) Segmented Executable Linker  Version 5.60.339 Dec  5 1994
Copyright (C) Microsoft Corp 1984-1993.  All rights reserved.

Object Modules [.obj]: new_target.obj
Run File [new_target.exe]: "new_target.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

c:\MASAM>LINK new_target.obj

Microsoft (R) Segmented Executable Linker  Version 5.60.339 Dec  5 1994
Copyright (C) Microsoft Corp 1984-1993.  All rights reserved.

Run File [new_target.exe]:
List File [nul.map]:
Libraries [.lib]:
Definitions File [nul.def]:

c:\MASAM>
c:\MASAM>
```

CONCLUSION

This project builds a mini compiler using the assembly language as mentioned above. It assembles the c code into assembly. We have also implemented assembly routines for decimal, hexadecimal, octal, binary etc. In this program instructions are converted into a machine-code or lower-level form so that they can be read and executed by the computer. This mini compiler includes the lexical analyser, the syntax analyser, the semantic analyser, accompanied by intermediate code generation with code optimisation phase. The implementation of our code generator ends with the code generation phase. The final code generated is the assembly language which is exhibited at the end user side.This compiler involves all the basic functions of C programming.

REFERENCES

(1)https://www.codeproject.com/Articles/30176/Mini-C-Compiler

(2) https://www.stack.nl/~marcov/compiler.pdf

(3)
https://books.google.co.in/books?id=A3yqQuLW5RsC&printsec=frontcover&dq=how+to+make+a+mini+compiler&hl=en&sa=X&ved=0ahUKEwjKx93MgJ3XAhWBpY8KHapbD7IQ6AEIKzAB#v=onepage&q&f=false

(4)
https://books.google.co.in/books?id=mdUSZudsBn0C&pg=PA50&dq=how+to+make+a+mini+compiler&hl=en&sa=X&ved=0ahUKEwjKx93MgJ3XAhWBpY8KHapbD7IQ6AEIUzAI#v=onepage&q=how%20to%20make%20a%20mini%20compiler&f=false

(5) http://www.wilfred.me.uk/blog/2014/08/27/baby-steps-to-a-c-compiler/