MINI PYTHON COMPILER

A PROJECT REPORT

# Submitted by

## BL.EN.U4AIE19007 Apoorva Mani BL.EN.U4AIE19041 Maturi Tanuj

BL.EN.U4AIE19068 Aishwarya V

# in partial fulfillment for the award of the degree of BACHELOR OF TECHNOLOGY

IN

## ARTIFICIAL INTELLIGENCE AND ENGINEERING



AMRITA SCHOOL OF ENGINEERING, BANGALORE AMRITA VISHWA VIDYAPEETHAM BANGALORE 560 035

June – 2022

### BONAFIDE CERTIFICATE

This is to certify that the project report entitled **MINI PYTHON COMPILER** submitted by

## BL.EN.U4AIE19007 Apoorva Mani BL.EN.U4AIE19041 Maturi Tanuj

BL.EN.U4AIE19068 Aishwarya V

in partial fulfillment of the requirements in the **Degree Bachelor of Technology** “Artificial Intelligence and Engineering” as part of 21AIE313 Introduction to Modern Compiler Design Project is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering, Bangalore.

|  |  |  |
| --- | --- | --- |
| <signature>  <Guide name> | <signature>  <Guide name> | Dr. Sriram Devanathan |
| <Designation> | <Designation> | Professor and Chairperson, |
| Dept. of CSE | Dept. of CSE | Dept. of CSE |

This project report was evaluated by us on ………

# ACKNOWLEDGEMENT

The satisfaction that accompanies successful completion of any task would be incomplete without mention of people who made it possible, and whose constant encouragement and guidance have been source of inspiration throughout the course of this project work.

We offer our sincere pranams at the lotus feet of **”AMMA”, MATA AMRITANANDAMAYI DEVI** who showered her blessing upon us throughout the course of this project work.

We owe our gratitude to **Dr. Manoj P.**, Director, Amrita School of Engineering, Bangalore.

We thank **Dr. Sriram Devanathan,** Principal and Chairperson-CSE, Amrita School of Engineering, Bangalore for his support and inspiration.

It is a great pleasure to express our gratitude and indebtedness to our project guide “**GUIDE NAME AND DESIGNATION”**, Department of Computer Science and Engineering, Amrita School of Engineering, Bangalore for her valuable guidance, encouragement, moral support, and affection throughout the project work.

We would like to thank express our gratitude to project panel members for their suggestions, encouragement, and moral support during the process of project work and all faculty members for their academic support. Finally, we are forever grateful to our parents, who have loved, supported, and encouraged us in all our endeavors.

# ABSTRACT

The Mini-Compiler contains all phases of compiler has been made for the language Python by using C language (till intermediate code optimization phase) and we used Python language itself for target code generation as well. The constructs that have been focused on are ‘if-else’ and ‘while’ statements. The optimizations handled for the intermediate code are ‘packing temporaries’ and ‘constant propagation’. Syntax and semantic errors have been handled and syntax error recovery has been implemented using Panic Mode Recovery in the lexer.

# TABLE OF CONTENTS

### Page no.

[ACKNOWLEDGEMENT i](#_TOC_250002)

[ABSTRACT ii](#_TOC_250001)

[CHAPTER 1- INTRODUCTION 6](#_TOC_250000)

CHAPTER 2 – REQUIREMENTS AND ANALYSIS 8

* 1. SOFTWARE REQUIREMENTS
  2. DIFFERENT MODULES OF PROJECT
  3. CONTEXT-FREE GRAMMAR

CHAPTER 3 – SYSTEM DESIGN 12

3.1 DESIGN STRATEGY

CHAPTER 4 – IMPLEMENTATION 14

* 1. MODULES USED WITH DESCRIPTION
  2. BUILD AND RUN THE PROGRAM

CHAPTER 5 – TESTING & RESULTS 16

CHAPTER 6 – CONCLUSION AND FUTURE ENHANCEMENT 22

REFERENCES 24

# CHAPTER - 1 INTRODUCTION

The Mini-Compiler contains all phases of compiler has been made for the language Python by using C language (till intermediate code optimization phase) and we used Python language itself for target code generation as well. The constructs that have been focused on are ‘if-else’ and ‘while’ statements. The optimizations handled for the intermediate code are ‘packing temporaries’ and ‘constant propagation’. Syntax and semantic errors have been handled and syntax error recovery has been implemented using Panic Mode Recovery in the lexer.

The screenshots of the sample input and target code output are as follows:

### Sample Input:

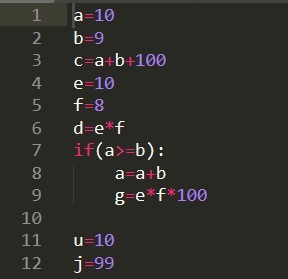


Figure 1 Sample Input python code

### Sample Output:

This the target code which is generated after ICG

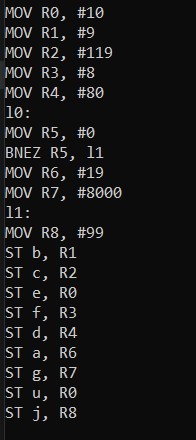


Figure 2 Sample output

# CHAPTER - 2 SYSTEM SPECIFICATIONS

* 1. **SOFTWARE REQUIREMENTS**

## ARCHITECTURE OF LANGUAGE

For this mini compiler, the following aspects of the Python language syntax havebeen covered:

* Constructs like ‘if-else’ and ‘while’ and the required indentation for these loops.
* Nested loops
* Integer and float data types

Specific error messages are displayed based on the type of error. Syntax errors are handled using the yyerror() function, while the semantic errors are handled by making a call to a function that searches for a particular identifier in the symbol table. The line number is displayed as part of the error message.

As a part of error recovery, panic mode recovery has been implemented for the lexer. It recovers from errors in variable declaration. In case of identifiers, whenthe name begins with a digit, the compiler neglects the digit and considers the rest as the identifier name.

Languages used to develop this project:

* C
* YACC
* LEX
* PYTHON

# DIFFERENT MODULES OF PROJECT

**Different Folders:**

1. **Token\_And Symbol\_Table**: This folder contains the code that outputs the tokens andthe symbol table.
2. **Abstract\_Syntax\_Tree**: This folder contains the code that displays the abstract syntaxtree.
3. **Intermediate\_Code\_Generation**: This folder contains the code that generates thesymbol table before optimisations and the intermediate code.

**Different Files:**

1. **proj.l**: It is the Lexical analyser file which defines all the terminals of the productions stated in the yacc file. It contains regular expressions.
2. **proj1.y**: Yacc file is where the productions for the conditional statements like if-elseand while and expressions are mentioned. This file also contains the semantic rules defined against every production necessary. Rules for producing three address code is also present.
3. **inp.py:** The input python code which will be parsed and checked for semantic correctness by executing the lex and yacc files along with it.

# CONTEXT-FREE GRAMMAR

REGEX

digits -> [0-9]

num -> digits+(\.digits+)?([Ee][+|-]?digits+)?

id -> [a-zA-Z][a-zA-Z0-9]\*

integer -> [0-9]+

string -> [a-z | A-Z | 0-9 | special]\*

special -> [ ! " # $ % & \ ( ) \* + , - . / : ; < = > ? @ [ \ \ ] ^ \_ ` { | } ~ ]

GRAMMAR

P ->S

S ->Simple S | Compound S | epsilon

Simple ->Assignment LB | Cond LB | Print LB | break | pass | continueAssignment -> id opassgn E1 | id opassgn cond | id listassgn Arr

| id strassgn Str

E1 -> E1 op1 E2 | E2

E2 -> E2 op2 E3 | E3

E3 -> E4 op3 E3 | E4

E4 -> num | id | (E1)

opassgn -> = | /= | \*= | += | -=op1 -> + | - op2 -> / | \*

op3 -> \*\*

LB -> \n

listassgn -> = strassgn -> = | += | -=

Arr -> [list] | [list] mul | [list] add | matmat -> [listnum] | [liststr]

list -> listnum | liststr | Range listnum -> num,listnum | epsilon | numliststr -

> Str,liststr | epsilon | Str

mul -> \* integer

add -> + Arr

Range -> range ( start , stop , step )start -> integer | epsilon stop -> integer

step -> integer | epsilon

Str -> string | string mul | string addstr

addstr -> + string

Compound -> if\_else LB | while\_loop LB

if\_else -> if condition : LB IND else | if condition : LB IND

| if condition : S | if condition : S else else -> else : LB IND | else : S

while\_loop -> while condition : LB IND | while condition : S

condition -> cond | (cond)

cond -> cond opor cond1 | cond1 cond1 -> cond1 opand cond2 | cond2cond2 -

> opnot cond2 | cond3

cond3 -> (cond) | relexp | bool

relexp -> relexp relop E1 | E1 | id | num relop -> < | > | <= | >= | == | != | in | not inbool -> True | False

opor -> || | or

opand -> && | and

opnot -> not | ~

IND -> indent S dedent

indent -> \t

dedent -> -\t

Print -> print ( toprint ) | print ( toprint,sep ) | print ( toprint,sep,end )

| print ( toprint,end ) toprint -> X | X,toprint | epsilonX -> Str | Arr | id

| num

sep -> sep = Str

end -> end = St

# CHAPTER - 3

**3.1 DESIGN STRATEGY**

1. **SYMBOL TABLE CREATION**

Linked list is being used to create the symbol table. The final output shows the label, value, scope, line number and type. We have created three functions to generate the symbol table. They are:

* Insert: It pushes the node onto the linked list.
* Display: It displays the symbol table.
* Search: It searches for a particular label in the linked list.

1. **ABSTRACT SYNTAX TREE**

This is being implemented using a structure that has three members whichhold the data, left pointer and right pointer respectively. The functions thataid in creating and displaying this tree are:

* BuildTree: It is used to create a node of this structure and add it to the existing tree.
* printTree: This function displays the abstract syntax tree using pre-order traversal.

1. **INTERMEDIATE CODE GENERATION**

We have used the stack data structure to generate the intermediate code that uses some functions, which are called based on some conditions.

1. **ERROR HANDLING**

* Syntax Error:

If the token returned does not satisfy the grammar, then yyerror() isused to display the syntax error along with the line number.

* Semantic Error:

If there is an identifier in the RHS of an assignment statement, the symbol table is searched for that variable. If the variable does not exist in the symbol table, this is identified as a semantic error and isdisplayed.

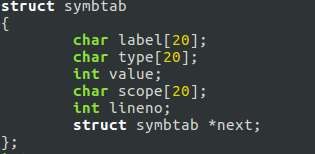
* Error Recovery:

Panic Mode Recovery is used as the error recovery technique, where if the variable declaration has been done with a number at the start,it ignores the number and considers the rest as the variable name. This has been implemented using regex.

# CHAPTER – 4 SYSTEM IMPLEMENTATION

* 1. **Modules used with description:**

1. SYMBOL TABLE CREATION

The following snapshot shows the structure declaration for symbol table:

*Figure 3 structure declaration for symbol table*

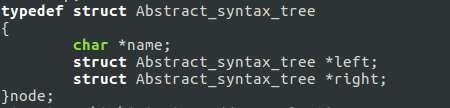
These are the functions used to generate the symbol table:

Figure 4 generate the symbol table

These snapshots are taken from proj1.y file in Token and Symbol table folder.

1. ABSTRACT SYNTAX TREE

The following data structure is used to represent the abstract syntax tree:



*Figure 5 abstract syntax tree*

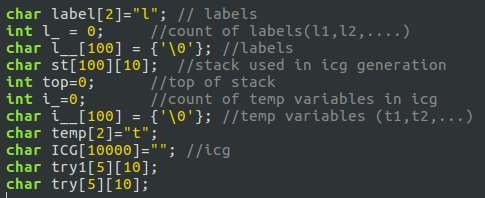
The following functions build and display the syntax tree:

Figure 6 display the syntax tree

These snapshots are taken from proj1.y file in Abstract syntax tree folder.

1. INTERMEDIATE CODE GENERATION

The following arrays act as stacks and are used for the generation ofintermediate code:



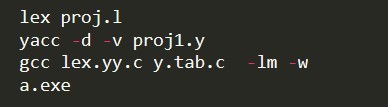
*Figure 7 intermediate code*

The following functions push onto the stack and generate the intermediatecode, when called based on various conditions:

Figure 8 intermediatecode, when called based on various conditions

### BUILD AND RUN THE PROGRAM:

The following screenshot displays what commands need to be executed to buildand run the program:



The above commands need to be executed on the terminal which is inside theproject folder that contains the code for the compiler.

# CHAPTER – 5

**SYSTEM TESTING AND RESULTS ANALYSIS**

## RESULTS AND SHORTCOMINGS

The mini-compiler built in this project works perfectly for the ‘if-else’ and ‘while’ constructs of Python language. Our compiler can be executed in differentphases by building and running the code separated in the various folders. The final code displays the output of all the phases on the terminal, one after the other. First, the tokens are displayed, followed by a ‘PARSE SUCCESSFUL’ message. Then abstract syntax tree is printed. Next, the symbol table along with the intermediate code is printed without optimisation. Finally, the symbol table andthe intermediate code after optimisation is displayed after the quadruples table. The final output is the target code, written in the instruction set architecture followed by the hypothetical machine model introduced in this project. This is for inputs with no errors. But in case of erroneous inputs, the token generation isstopped on error encounter and the corresponding error message is displayed.

This mini-compiler has the following shortcomings:

* User defined functions are not handled.
* Importing libraries and calling library functions is not taken care of.
* Datatypes other than integer and float, example strings, lists, tuples,dictionaries, etc have not been considered.
* Constructs other than ‘while’ and ‘if-else’ have not been added in thecompiler program.

SNAPSHOTS

# TEST CASE 1 (Correct input):

**Input:**

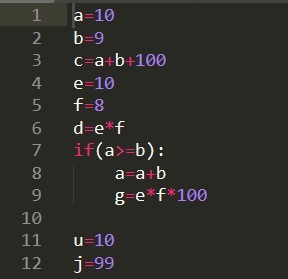
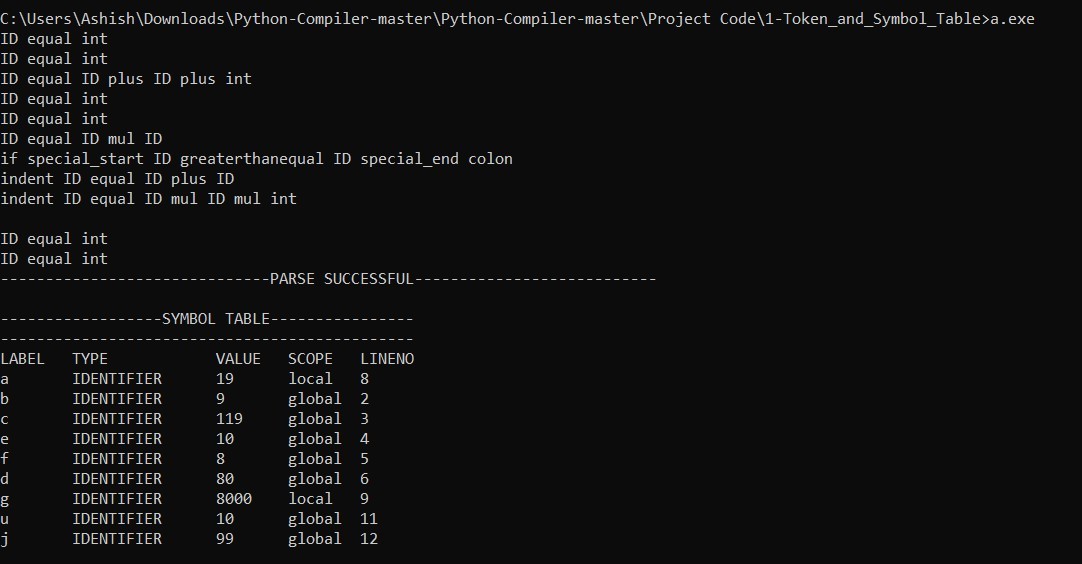


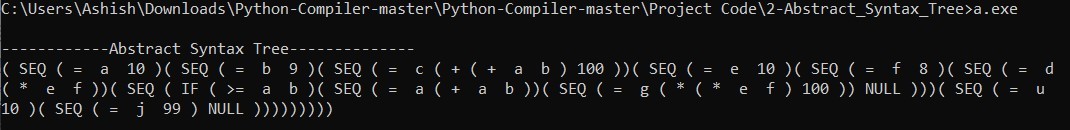
Figure 9 input code

### Tokens and Symbol Table:



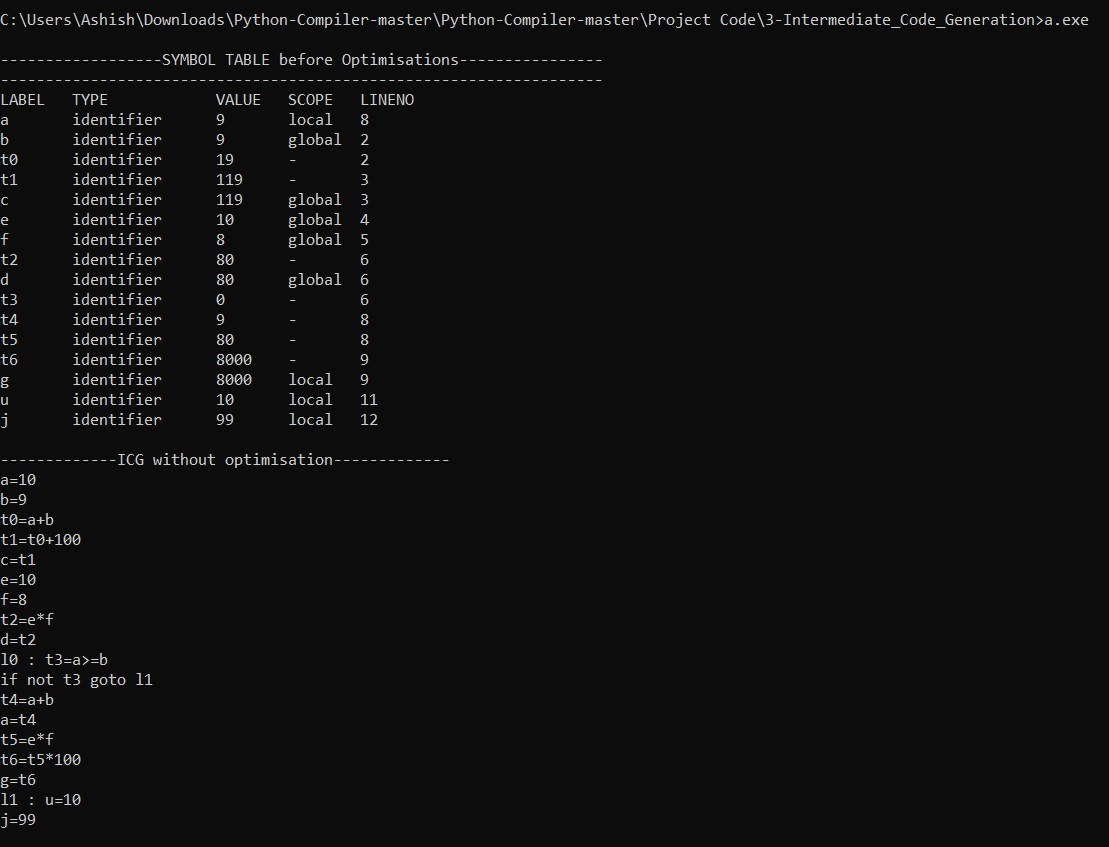
*Figure 10 case output symbol table*

**Abstract Syntax Tree:**



*Figure 11 abstract syntax tree*

### Symbol Table and Unoptimized Intermediate Code



*Figure 12 intermediate code output*

TEST CASE 2 (Syntax Error):

**Input:**

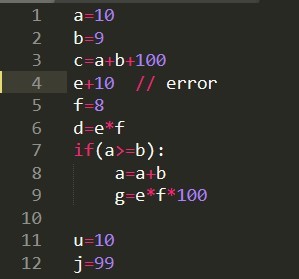
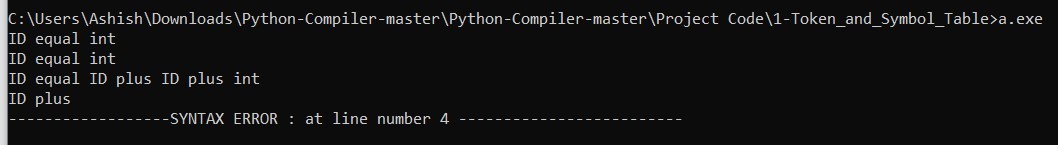


Figure 13 sample case input

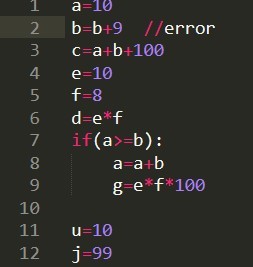
### Output:



*Figure 14 syntax error*

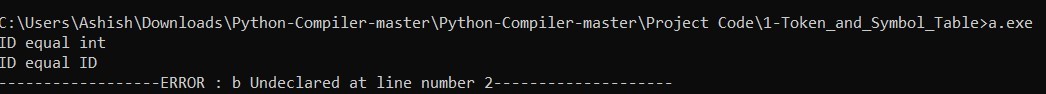
TEST CASE 3 (Semantic Error):

**Input:**



*Figure 15 sample case 3 input*

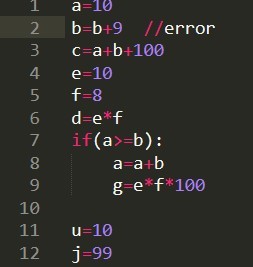
### Output:



*Figure 16 sample case 3 output*

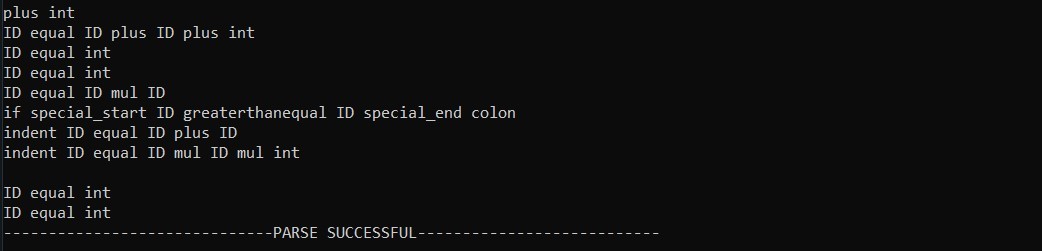
TEST CASE 4 (Error Recovery):

**Input:**



*Figure 17 case 4 input*

### Output:



*Figure 18 sample output*

<Chapter Name> <Month, year>

# CHAPTER – 6 CONCLUSION AND FUTURE SCOPE

## CONCLUSIONS

* + Making a full complete compiler is a very difficult task and it takes lots of time to make it. So, we have successfully made a minicompiler which performs following operations:
    1. This is a mini compiler for python using lex and yacc files which takes in a python program and according to the context free grammar written, the program is validated.
    2. Regular Expressions are written to generate the tokens.
    3. Symbol table is created to store the information about the identifiers.
    4. Abstract syntax tree is generated and displayed according to the pre-order tree traversal.
    5. Intermediate code is generated, and the data structure used foroptimization is Quadruples. The optimization techniques used are constant propagation and packing temporaries.

# FUTURE ENHANCEMENTS

This mini compiler can be enhanced to a complete compiler for the Python language by making a few improvements. User defined functions can be handled and the functionality of importing libraries and calling library functions can be taken care of. Datatypes other than integer, example strings, lists, tuples, dictionaries, etc. can be included and constructs other than ‘while’ and ‘if-else’, like ‘for’ can be added in the compiler program. The output can be made to lookmore enhanced and beautiful. The overall efficiency and speed of the program can be improved by using some other data structures, functions, or approaches.

# REFERENCES

* + - 1. **Lex and Yacc**

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

* + - 1. **Introduction about Flex and Bison**

<http://dinosaur.compilertools.net/>

* + - 1. **Full Grammar Specification**

https://docs.python.org/3/reference/grammar.html

* + - 1. **Introduction to Yacc**

https://[www.inf.unibz.it/~artale/Compiler/intro-yacc.pdf](http://www.inf.unibz.it/~artale/Compiler/intro-yacc.pdf)

### Intermediate Code Generation

https://2k8618.blogspot.com/2011/06/intermediate-code-generator-

for.html?m=0

### A Code Generator to translate Three-Address intermediate code to MIPS assembly code

https://www2.cs.arizona.edu/~debray/Teaching/CSc453/DOCS/3addr2spi m.pdf