CSB353 : COMPILER DESIGN

MINI PYTHON COMPILER SYNOPSIS
(ZERCOIN)

Submitted By:

Vadlamudi Neel Vittal Bharath (191210053)
Vinay Choudhary (191210059)
Vinay Jaiswal (191210060)
Branch: CSE
Semester: 6th
Group: 2

Submitted To:

Dr. Shelly Sachdeva

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

TABLE OF CONTENTS

INTRODUCTION

The Mini-Compiler will contain all phases of compiler and compiler will be made for the language Python by using C language (till intermediate code optimisation phase) and we will use Python language itself for target code generation as well . The constructs that will be focused on are ‘if-else’ and ‘while’ statements. It will handle Syntax and semantic errors.

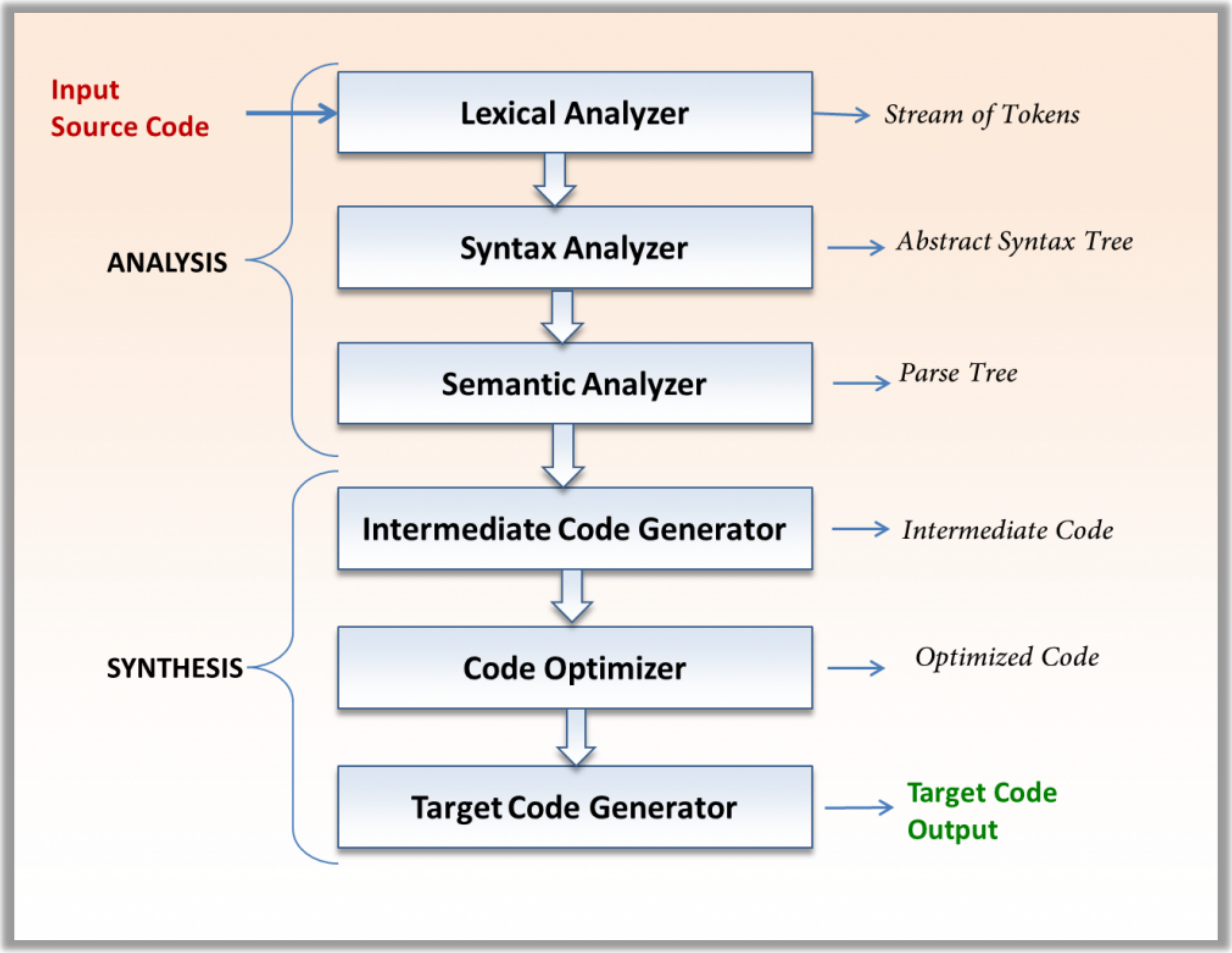
FEATURE

For this mini-compiler, the following aspects of the Python language syntax will be covering:

- 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.

THE PHASES OF COMPILER DESIGN



Lexical Analysis

Lexical Analysis is the first phase when compiler scans the source code. This process can be left to right, character by character, and group these characters into tokens.
Here, the character stream from the source program is grouped in meaningful sequences by identifying the tokens. It makes the entry of the corresponding tickets into the symbol table and passes that token to the next phase.

Syntax Analysis

Syntax analysis is all about discovering structure in code. It determines whether or not a text follows the expected format. The main aim of this phase is to make sure that the source code written by the programmer is correct or not.
Syntax analysis is based on the rules based on the specific programming language by constructing the parse tree with the help of tokens. It also determines the structure of source language and grammar or syntax of the language.

Semantic Analysis

Semantic analysis checks the semantic consistency of the code. It uses the syntax tree of the previous phase along with the symbol table to verify that the given source code is semantically consistent. It also checks whether the code is conveying an appropriate meaning.
Semantic Analyzer will check for Type mismatches, incompatible operands, a function called with improper arguments, an undeclared variable, etc.

Intermediate Code Generation

Once the semantic analysis phase is over the compiler generates intermediate code for the target machine. It represents a program for some abstract machine.
Intermediate code is between the high-level and machine level language. This intermediate code needs to be generated in such a manner that makes it easy to translate it into the target machine code.

Code Optimization

The next phase is code optimization or Intermediate code. This phase removes unnecessary code lines and arranges the sequence of statements to speed up the execution of the program without wasting resources. The main goal of this phase is to improve on the intermediate code to generate a code that runs faster and occupies less space.

Code Generation

Code generation is the last and final phase of a compiler. It gets inputs from code optimization phases and produces the page code or object code as a result. The objective of this phase is to allocate storage and generate relocatable machine code.
It also allocates memory locations for the variable. The instructions in the intermediate code are converted into machine instructions. This phase converts the optimized or intermediate code into the target language.
The target language is machine code. Therefore, all the memory locations and registers are also selected and allotted during this phase. The code generated by this phase is executed to take inputs and generate expected outputs.

DESIGN PROCEDURE

- 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.
- Regular Expressions are written to generate the tokens.
- Symbol table is created to store the information about the identifiers.
- Abstract syntax tree is generated and displayed according to the pre-order tree traversal.
- Intermediate code is generated, and the data structure used for optimisation is Quadruples. The optimisation techniques used are constant propagation and packing temporaries.
- The optimized intermediate code is then converted to the Target code using a hypothetical machine model.

USED LANGUAGES

Languages we will use to develop this project:

- C
- YACC
- LEX
- PYTHON

SOFTWARE AND HARDWARE REQUIREMENTS

Software to be used:

Lex tool
Turbo c/gcc
Yacc parser tool

Hardware to be used:

128 mb RAM
64 bit machine
800 mHz