Project Report

CMPN403: Languages and Compilers

Submitted To: Dr. Ayman AboElhassan

Team 8

Name
Abdullah Ayman
Hussein Mostafa
Omar Sherif
Salah Mohamed

Table of Contents

Project Overview

The objective of this project was to design and implement a simple programming language using Lex and Yacc, focusing on core compiler design principles. This project involved constructing a lexer, a parser, and an intermediate code generator that produces quadruples, which serve as the foundation for translating high-level language constructs into low-level assembly-like instructions.

The language supports fundamental programming constructs, including variable declarations, arithmetic expressions, relational operations, conditional statements (`if-else, switch case`), loops (`while, repeat until, for`), and Boolean logic. Throughout the project, emphasis was placed on ensuring correctness, efficient intermediate representation, and modularity.

The main components of the project included:

1. Lexical Analysis:
    - A lexer was implemented using Lex to tokenize the input source code.
    - Tokens such as identifiers, keywords, operators, and literals were defined to facilitate syntax analysis.
    - Special handling was implemented for multi-line comments, whitespace, and error reporting to improve usability.
2. Syntax Analysis:

    A parser was built using Yacc to validate the syntax of the input program based on the language's grammar.
    - The parser leveraged an LL(1) grammar and incorporated mechanisms to produce meaningful error messages for incorrect syntax.
3. Intermediate Code Generation:
    - Quadruples were used as the intermediate representation (IR) of the program. Each quadruple captures a single operation, its operands, and the result, allowing for language-independent analysis and optimization.
    - Constructs such as assignments, arithmetic expressions, and control flow structures (`if-else` and `while`) were translated into quadruples to enable straightforward conversion to assembly.

4. Control Flow:

   Conditional statements and loops were implemented using temporary variables and labels for branch and jump operations.

   - Logical operations and relational expressions were evaluated with support for short-circuit evaluation.

5. Modular Design:

   - Each phase of the compilation process was implemented in a modular manner, with clear separation of concerns between lexical, syntactic, and semantic processing.
   - Temporary variables and labels were systematically generated to handle complex expressions and branching efficiently.

This project provided hands-on experience in compiler construction and served as a practical application of theoretical concepts such as parsing, semantic analysis, and intermediate code generation. The final deliverable is a fully functional compiler for the designed language, capable of producing accurate intermediate representations for various programming constructs.

Tools & technologies:

- g++11
- Bison
- Flex
- Tkinter for GUI
- CustomTkinter for GUI

Tokens

Numeric and Value Tokens

Token	Description	Example
INTEGER	Matches whole numbers	123, 0
FLOATING	Matches decimal numbers	123.45
CHARACTER	Single character in quotes	‘a’
CHARARRAY	String in double quotes	“hello”
BOOLEAN	True or False values	True, False

Keywords:

Token	Description	Example
WHILE	Loop control	“while”
REPEAT	Do-while style loop	“repeat”
UNTIL	Do-while style loop	“until”
FOR	For loop control	“for”
SWITCH	Switch statement control	“switch”
CASE	Switch statement control	“case”
IF	Conditional control	“if”
THEN	Conditional control	“then”
ELSE	Conditional control	“else”
FUNCTION	Function definition and control	“function”

RETURN	Function definition and control	“return”
VOID	Function return type	“void”

Data Types:

Token	Description	Example
INT	Integer type declaration	“int”
FLOAT	Floating-point type declaration	“float”
BOOL	Boolean type declaration	“bool”
CHAR	Character type declaration	“char”
STRING	String type declaration	“string’
CONST	Constant variable modifier	“const”

Operators:

Token	Description	Example
Arithmetic	+, -, *, /, ^	

Comparison	>=, <=, ==, !=, <, <	
Logical	&&, ||	
Assignment	=	

Other:

Token	Description	Example
VARIABLE	Identifiers (variable, function names)	
Delimiters	(){}[];,	
Comments	//, /* */	

Quadruples

Quadruple	Description	Example (op, arg1, arg2, res)
ASSIGN	Assigns a value to a variable	
ADD	Addition operation	
SUB	Subtraction operation	
MUL	Multiplication operation	
DIV	Division operation	
POW	Power/exponentiation operation	
JMP	Unconditional jump to a label	
JF	Jump if false (conditional branch)	
label	Defines a jump target	
PUSH	Push parameter for function call	
POP	Pop return value or parameter	
EQ	Equal comparison	
NE	Not equal	
LT	Less than comparison	
GT	Greater than	
LE	Less or equal	
GE	Greater or equal	
AND	Logical AND	

OR	Logical OR	
NOT	Logical NOT	

Workload Division

Abdullah Ayman	Quadruples, SymbolTable
Hussein Mostafa	Quadruples, SymbolTable
Omar Sherif	Quadruples, SymbolTable
Salah Mohamed	Quadruples, SymbolTable