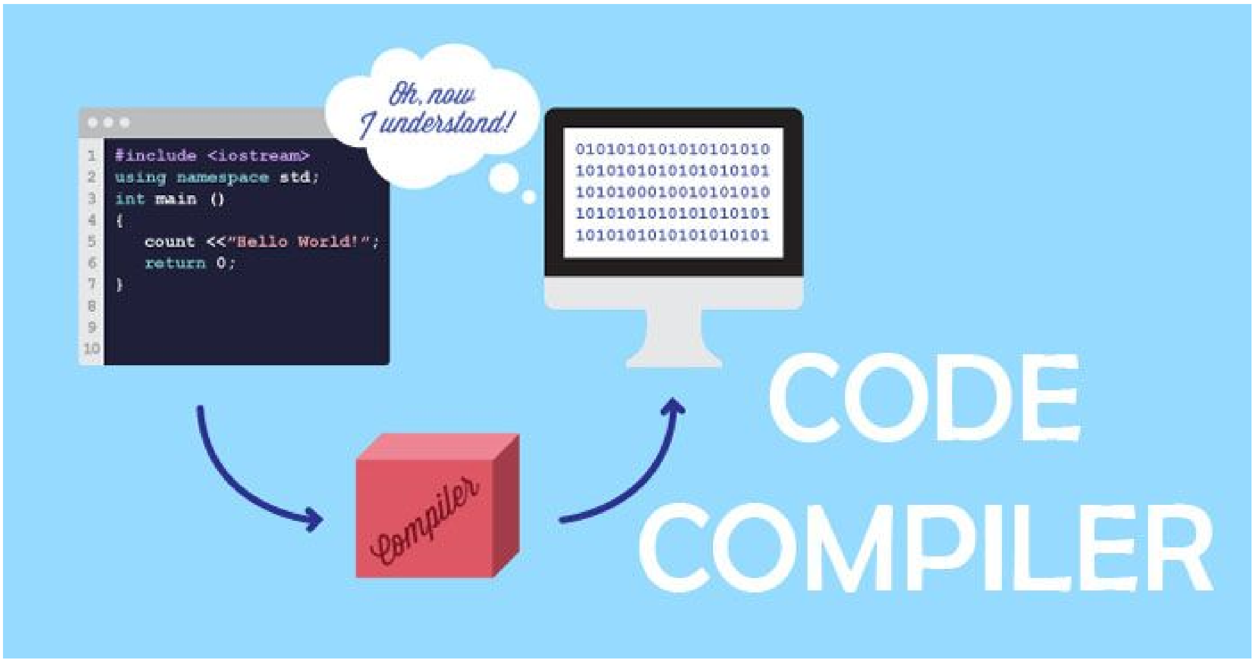

Compilers Project Report

Antoun Wagdy
Malak Fahim
Mohamed Atef
MennaTullah Mustafa

Table of Contents

Project Overview

This project introduces a simple compiler for a tiny-C like programming language.

Supported Functionalities

- Declaring variables and constants of 3 type: int, float, string.
- Left-associative mathematical expressions with the precedence as follows (decreasingly):
    - Parenthesis.
    - Multiplication and Division.
    - Addition and Subtraction.
- Left-associative logical expressions with the support to:
    - Parenthesis as the highest precedence.
    - “&” and “|” operators with the minimum checks required.
    - Other logical expressions.
- Assignment statement.
- Controlling statements such as:
    - If-then.
    - If-then-else.
    - For loop.
    - While loop.
    - Repeat-Until loop.
    - Switch-case.
- Nested blocks.
- Functions conditioned on that there is a “main” function.
- Syntax errors handling with declarative messages.

Semantic Error Handling

- Multiple declarations of the same variable, constant or function.
- Assessing that the LHS and RHS of any arithmetic or logical expressions are **int** or **float** variables or constants.
- Assessing that LHS and RHS of assignment statement are compatible.
- **Allowing** expressions that contains combination of int and float variables or constants.
- Adding **type conversion** quadrables if needed.
- Allowing **dynamic up-casting** with the arithmetic expressions.
- Allowing **dynamic down-casting** with the assignment statements.

Tools and Technologies Used

1. Flex 2.5.4a (fast lexical analyzer generator).
2. Bison 2.4.1 (Yacc-compatible parser generator).
3. GCC.

Tokens

Keywords

Token	Keyword it captures
WHILE	“while”
FOR	“for”
REPEAT	“repeat”
UNTIL	“until”
SWITCH	“switch”
CASE	“case”
DEFAULT	“default”
IF	“if”
ELSE	“else”
PRINT	“print”
DEF	“def”
AS	“as”
VAR	“var”
CONST	“const”
INT	“int”
FLOAT	“float”
STRING	“string”
FUN	“function”
CALL	“call”

Token	Keyword it captures
NE	“!=”
EQ	“==”
GE	“>=”
LE	“<=”

Other Tokens

Token	What it captures
INTNUM	The integer numbers (5, 57, 0, 8735).
FLOATNUM	The floating-point numbers (0.5, 5.7, 0.547, 87.35).
QUOTESTRING	The strings entered between double quotes (“Hi”, “Mohamed Atef”).
IDENTIFIER	The allowed identifiers names. It must start with a letter followed optionally by a combination of letters and the “_”.
BODY	All the function’s body in one node in the parse tree.
FUNCTIONS	All the functions declared in the program.
IFX	The precedence of if-else.

language production rules

Program

- **Program → functions**

Functions

- **Functions → functions function | Epsilon**

Function

- **Function → DEF IDENTIFIER AS FUN '{' fun_body '}'**

Fun_body

- **Fun_body → fun_body stmt | fun_body err_stmt |Epsilon**

err_stmt

- **err_stmt → error ';' | error '}' | error ')' | error REPEAT**

default_stmt

- **defult_stmt → DEFAULT stmt**

case_stmt

- **Case_stmt → CASE '(' expr ')' stmt | '{' case_list '}' | '{' case_list defult_stmt '}'**

case_list

- **case_list → case_stmt | case_list case_stmt**

logic_expr

- logic_expr → expr '<' expr
  | expr '>' expr
  | expr GE expr
  | expr LE expr
  | expr NE expr
  | expr EQ expr
  | '(' logic_list ')'

logic_list

- logic_list → logic_expr | logic_list '|' logic_expr | logic_list '&' logic_expr

stmt

- stmt → DEF IDENTIFIER AS INT VAR '=' expr ';'
  | DEF IDENTIFIER AS FLOAT VAR '=' expr ';'
  | DEF IDENTIFIER AS STRING VAR '=' expr ';'
  | DEF IDENTIFIER AS INT VAR ';'
  | DEF IDENTIFIER AS FLOAT VAR ';'
  | DEF IDENTIFIER AS STRING VAR ';'
  | DEF IDENTIFIER AS INT CONST '=' expr ';'
  | DEF IDENTIFIER AS FLOAT CONST '=' expr ';'
  | DEF IDENTIFIER AS STRING CONST '=' expr ';'
  | ';'
  | PRINT expr ';'
  | IDENTIFIER '=' expr ';'
  | WHILE '(' logic_list ')' stmt
  | FOR '(' stmt logic_list ';' stmt ')' stmt
  | REPEAT stmt UNTIL '(' logic_list ')' ';'
  | SWITCH '(' expr ')' case_stmt
  | IF '(' logic_list ')' stmt
  | IF '(' logic_list ')' stmt ELSE stmt
  | '{' stmt_list '}'
  | CALL IDENTIFIER ';'

stmt_list

- stmt_list → stmt | stmt_list stmt

expr

- **expr → INTNUM**
  | FLOATNUM
  | QUOTESTRING
  | IDENTIFIER
  | expr '+' expr
  | expr '-' expr
  | expr '*' expr
  | expr '/' expr
  | '(' expr ')'

Quadruples Description

Quadruple	Description
<variable name> DD	Reserves double-word memory location to this variable.
<variable name> DQ	Reserves quad-word memory location to this variable.
<variable name> Times <number> DB	Reserves <number> * single-byte memory location to this variable.
pushS <variable name>	Push the string variable to the top of the stack until reaching a terminating letter (\0).
pushD <variable name>	Push the double-word variable to the top of the stack.
pushQ <variable name>	Push the quad-word variable to the top of the stack.
popS <variable name>	Pop a string from the top of the stack until reaching a terminating letter (\0) into the variable.
popD <variable name>	Pop double-word variable from the top of the stack into the variable.
popQ <variable name>	Pop quad-word variable from the top of the stack into the variable.
printS	Pop a string from the top of the stack until reaching a terminating letter (\0) and print it.
printD	Pop double-word variable from the top of the stack and print it.
printQ	Pop quad-word variable from the top of the stack and print it.
convDQ	Extend the double-word on the top of the stack to become quad-word.
convQD	Shrink the quad-word on the top of the stack to become double-word.
addD	Add the first 2 double-words on the top of the stack

Quadruple	Description
addQ	Add the first 2 quad-words on the top of the stack
subD	Subtract the first 2 double-words on the top of the stack
subQ	Subtract the first 2 quad-words on the top of the stack
mulD	Multiply the first 2 double-words on the top of the stack
mulQ	Multiply the first 2 quad-words on the top of the stack
divD	Divide the first 2 double-words on the top of the stack
divQ	Divide the first 2 double-words on the top of the stack
compDEQ	Reset the Z-flag if the first 2 double-words on the top of the stack are equal, set it otherwise.
compQEQ	Reset the Z-flag if the first 2 quad-words on the top of the stack are equal, set it otherwise.
compDNE	Reset the Z-flag if the first 2 double-words on the top of the stack are not equal, set it otherwise.
compQNE	Reset the Z-flag if the first 2 quad-words on the top of the stack are not equal, set it otherwise.
compDGT	Reset the Z-flag if the second double-word on the top of the stack is greater than the first one, set it otherwise.
compQGT	Reset the Z-flag if the second quad-word on the top of the stack is greater than the first one, set it otherwise.
compDLT	Reset the Z-flag if the second double-word on the top of the stack is less than the first one, set it otherwise.
compQLT	Reset the Z-flag if the second quad-word on the top of the stack is less than the first one, set it otherwise.
compDGE	Reset the Z-flag if the second double-word on the top of the stack is greater than or equal to the first one, set it otherwise.
compQGE	Reset the Z-flag if the second quad-word on the top of the stack is greater than or equal to the first one, set it otherwise.
compDLE	Reset the Z-flag if the second double-word on the top of the stack is less than or equal to the first one, set it otherwise.

Quadruple	Description
compQLE	Reset the Z-flag if the second quad-word on the top of the stack is less than or equal to the first one, set it otherwise.
jz <label>	Jmp to the label if the Z-flag is set.
jnz <label>	Jmp to the label if the Z-flag is not set.
Jmp <label>	Jmp to the label.
Call <function name>	Call the function.
ret	Return from procedure.
exit	Finish executing.