Indian Institute of Technology, Kanpur

Project Report, CS335

A C Compiler

Vipin Chhillar(150805), vipinc@iitk.ac.in

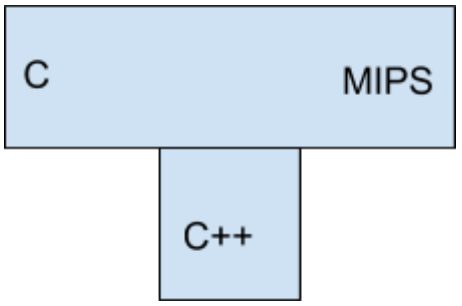
Anuj Chauhan(150119), anujc@iitk.ac.in

16 April 2019

OVERVIEW

Implemented a toy C Compiler by taking adequate minimal features of the real world C compiler like - Basic Arithmetic Operations, Boolean Expressions, Control Structure like IF and IF-ELSE, Basic Loops like FOR, WHILE and DO-WHILE and Functions. We've used **LEX-YACC**[1] as provided in CPP for the implementation of LEXER and PARSER and used **GRAPHVIZ**[2] in Milestone 1, rest of the part is done without using any platform. There are four MILESTONES in the compiler construction. Each Milestone has helped to finally build the complete compiler.

T DIAGRAM



- **Source Language : C**[3]
- **Implementation Language: C++**
- **Target Language: MIPS**

FEATURES SUPPORTED

- **Data Types:** 'int' only.
- **Operators (int):**
  - Arithmetic **+, -, /, *, %**
  - Relational **<, >, <=, >=, ==, !=**
  - Logical ops **&&,** ||
  - Assignment **=**

- **Boolean Expressions:**
- **If-else:**
- **Loops:**
    - For:
    - While:
    - Do-While:
- **Functions and Func Calls:**

FEATURES NOT SUPPORTED

Float and Double types.
Post/Pre increment/decrement.
Structs and Enums.
Pointers.
Comments.
Switch Case.
One and multidimensional arrays.
Function Overloading.

IMPLEMENTATION

- Milestone 0 (LEXER):
    - Lexer takes the source file as input and generates the string of tokens and pass these token to the parser which further processes the tokens.

- Milestone 1 (PARSER):
    - Generate a AST using Graphviz by implementing some functions like:
    - **makeNode():** for making a new node to enter in the AST.
    - **makeChild():** for making edges between two nodes in AST.

- Milestone 2 (IR and SYMBOL TABLE):
    - Symbol Table Data Structure:
        - One Global ST which contains all functions and their arguments, and pointers to the local STs.

        - One local ST for every function which contains all the variables and their types.
    - IR
        - X = y op Z — op = +, -, *, / : operation instruction
        - X=Y : copy instructions
        - Goto L — (L is a Label) : unconditional jump
        - If (X relop Y) goto L : conditional jump
        - If (X) goto L — X = true /false : conditional jump
        - foo x y z : function declaration.
        - Int x : variable declaration.
        - X = call foo a b c : function call (With Return).
        - Call foo a b c : function call (Without Return).
        - Return : Function return.
        - Return x : Function returning variable ‘x’.
        - Exit : Exit Call.

- The Final Destination (ASSEMBLY CODE):
    - Algorithm for converting IR Code to assembly:
        - Identified the basic blocks by implementing a function named **findBasicBlock().**
        - Calculated the Next Use information for every basic block using **nextUse()**
        - Generated code for every basic block while implementing **Register Spilling with 4 Registers.**
        - Used the functions **getReg()** for implementing Register Spilling heuristics.
        - Mapped every **IR instruction to MIPS instruction**.

References:

[1] Implementation of a simple Calculator using lex and yacc.
[2] Dot language used in milestone 1.
[3] Grammar taken from ANSII C