CO-351 Compiler Design Lab

Compiler for C Language Project

Arvind Ramachandran 15CO111
Aswanth P P 15CO112

Abstract

Our aim is to design a compiler for subset of C Programming Language. We are trying to implement the following features

Modules

1. Lexical Analyzer
2. Syntax Analyzer
3. Semantic Analyzer
4. Intermediate Code Generator

1. Lexical Analyzer

- Identification of Keywords, Identifiers, Operators (Relational, Logical and Arithmetic), Punctuators, Constants (Integer, Character and Float) and String Literals with invalid string error handling
- Single and Multi Line Comments with error handling
- Data Types (int, float and char) with modifiers (unsigned and signed) and types (short,long).
- Procedures with return type int,float and char
- Parenthesis matching with error reporting.
- Invalid character identification and error reporting

2. Syntax Analyzer

In this phase syntax analyzers receive their inputs, in the form of tokens, from lexical analyzers.Checks the expression from this token are syntactically correct. We are trying to implement following features.

- Syntax Checking for arithmetic , logical and relational expressions
- Productions rules for control statements such as if,if else and nested if statements

- Production rules for unary plus and minus
- Production rule and syntax checking for ‘for’ loop and nested for loop
- Syntax checking for function declaration and parameter passing
- Array indexing syntax errors
- Scanf and printf syntax errors
- Missing semicolon and unbalanced parenthesis

3. Semantic Analyzer

In this phase, we extract necessary semantic information from the source code which is impossible to detect in parsing. We are trying to implement following features

- Scope of the identifiers declared
- Duplicate declaration of identifiers
- Function declaration and its scope
- Actual and formal parameter matching(number and type of parameters)
- Matching function return type
- Checking array indexing with in the given bound
- Invalid array indexing while declaration (array limit less than one)
- Type mismatch of variables

4. Intermediate Code Generator

In this phase , We are trying to generate language independent three-address code for given source program which is lexically,syntactically and semantically correct .

It receives input in the form of annotated syntax tree That syntax tree then can be converted into a linear representation, (postfix notation).Intermediate code tends to be machine independent code. Therefore, code generator assumes to have unlimited number of memory storage (register) to generate code.This three-address code can be converted to MIPS assembly code .