Lab Assignment 10

Intermediate representation(s)/ generating Intermediate representation(s)

Q1 [AST]. There are various intermediate representations such as Abstract-Syntax-Tree (AST), Directed-acyclic-graph (DAG), and 3-address code. Consider your expression grammar from assignment 9 with operations such as addition, subtraction, multiplication and division.

1. Using semantic actions, design S-attributed/L-attributed translation grammar to generate **AST**. Explain all the semantic actions that you considered. Submit a document with brief explanations.
2. Implement your solution (NOTE: We had discussed about combining evaluation of semantic actions with top-down/bottom-up parsing). Your program should take an expression as input and generate its corresponding **AST** as output (you may decide about how you present/ store the output).

Q2 [3-address code]. There are various intermediate representations such as Abstract-Syntax-Tree (AST), Directed-acyclic-graph (DAG), and 3-address code. Consider your expression grammar from assignment 9 with operations such as addition, subtraction, multiplication and division.

1. Using semantic actions, design S-attributed/L-attributed translation grammar to generate -**3-address code**. Explain all the semantic actions that you considered. Submit a document with brief explanations.
2. Implement your solution (NOTE: We had discussed about combining evaluation of semantic actions with top-down/bottom-up parsing). Your program should take an expression as input and generate its corresponding **3-address code** as output.

Q3 [**DAG**- ***OPTIONAL***]. Semantic actions for generating **DAG** for the above problem. The output should be DAG of the input expression in a readable form.

Q4 [***OPTIONAL***]. GCC intermediate codes:

You may explore the internals, and understand the intermediate representations used by gcc.

Produce internal representations for some simple programs (program with few simple basic statements, program with 1 for loop etc), observe some of the intermediate representations generated by gcc for the considered programs.

The GCC compiler uses three intermediate representations:

3. **GENERIC**: language independent tree representation of the entire function
4. **GIMPLE**: three-address representation generated from GENERIC
5. **RTL**: low-level representation known as register transfer language

gcc produces the textual forms of the following intermediate representations of a program being compiled:

1. Abstract Syntax Tree (AST). The -fdump-tree-original-raw switch dumps the textual representation of the AST for given input source.
2. Gnu SIMPLE representation (GIMPLE). The -fdump-tree-gimple-raw switch dumps the GIMPLE representation of the input source (more readable form without -raw).

3. Control Flow Graph (CFG). The -fdump-tree-cfg-raw switch dumps the CFG form of the GIMPLE code.
4. Register Transfer Language (RTL IR). The -da switch dumps the RTL IR of the input source program with the pass number as a part of the dump file name.