Thomas Dooms
Ward Gauderis
2de bachelor Informatica
Compilers

C Compiler

Verslag 1

In overleg met Prof. Perez hebben we besloten onze C compiler in C++ te schrijven vermits zowel Antlr als LLVM C++ runtimes hebben. We hebben exact dezelfde setup zoals beschreven staat in de opdrachten, maar dan niet in Python.

Huidige Functionaliteit

Opdracht 1:

Verplicht:

- Binary operations: +, -, *, /
- Binary operations: <, >, ==
- Unary operators: +, -
- Brackets
- AST en dot visualisatie

Optioneel:

- Binary operator: %
- Comparison operators: >=, <=, !=
- Logical operators: &&, ||, ! (nog niet representeerbaar in LLVM IR omwille van short-circuit evaluation)
- Constant folding

Extra:

- Hexadecimale, octale en binaire integer literals
- CST visualisatie

Opdracht 2:

Verplicht:

- Types: char, float, int, pointer
- Reserved words: const
- Variables: declaration, definition, assignment, identifiers
- AST en dot visualisatie
- Error analyse: syntax, semantic errors

Optioneel:

- Unary operators: ++, -- (zowel prefix als postfix)
- Conversions: conversion operator, warnings
- Constant propagation

Extra:

- Float literals in wetenschappelijke notatie

Opdracht 3:

Verplicht:

- Comments: single line, multiline
- Printf
- AST en dot visualisatie
- LLVM code generation

Optioneel:

- Comments in LLVM IR hebben we uiteindelijk uit het project verwijderd omdat het erg moeilijk is deze op de juiste plek in de IR te plaatsen door mogelijke optimisaties. Daarbij komt dat het niet mogelijk is met Antlr om comments binnen expressies te parsen (wat wel gaat in gcc).

Extra:

- Printf kan ook r-values die het resultaat zijn van expressies printen.

Grammatica

In de grammatica is elke statement voorlopig een declaration, expression of printf-statement. De expression is op zo'n manier opgesteld dat de volgorde van bewerkingen gerespecteerd wordt zonder gebruik te moeten maken van left-recursion (wat LL parsers niet aankunnen). Dit zorgt voor een zeer grote CST (zie de CST in dot), maar wordt vereenvoudigd in de omzetting naar de AST (zie AST in dot).

AST

Constant folding en propagation gebeuren via virtuele functies in onze compiler. Elk node in de AST kan zichzelf folden zodat deze operaties recursief zijn gedefinieerd. Als bv. een binaire plus operator merkt dat zijn beide operands gefold kunnen worden at compile time, foldt de operation zelf ook. Het basisgeval van deze recursieve operatie is de literal node. Constant Propagation is in deze zin gewoon de folding van een constante variabele in de AST. Door deze op dezelfde manier te folden, kunnen we door middel van één recursieve functie de AST minimaliseren.

LLVM

Zoals eerder vermeld, is alle functionaliteit (buiten logical operators vanwege short-circuit evaluation en comments vanwege optimalisaties) ook geïmplementeerd in LLVM IR. Elke node in de AST heeft een functie die met behulp van de LLVM runtime-libraries IR-code genereert. Deze IR wordt dan geprint naar een 'll'-bestand. Hierbij gebeuren geen optimalisaties. Het is echter mogelijk de LLVM optimalisatiepasses aan te zetten via een CLI flag.

Error Handling

Er wordt een onderscheid gemaakt tussen 3 soorten exceptions: syntax errors, semantic errors en internal errors. Syntax errors worden voornamelijk opgemerkt tijdens het parsen met Antlr, semantic errors worden ontdekt in de AST en internal errors duiden aan dat bepaalde functionaliteit nog niet

aanwezig is. Hier proberen we zo dicht mogelijk aan te leunen bij gcc zodat IDE's zoals Clion ook gebruik kunnen maken van de exception output.

Testing

Voornamelijk voor de errorhandling en IR generation hebben we veel tests geschreven. De uitkomsten hebben we steeds vergeleken met gcc zodat we zo dicht mogelijk bij de referentie zitten. Zie het volgende punt voor enkele voorbeelden van speciale randgevallen die we zo ontdekt hebben. Zover wij hebben kunnen testen, geeft gcc enkel errors of warnings indien wij dat ook doen.

Randgevallen

Unary Expressions:

- +PtrType, -PtrType zijn niet toegestaan

Binary Expressions:

- PtrType – IntType is toegestaan maar IntType – PtrType niet
- Eender welke operation met FloatType en Ptr (buiten logical operators) is niet toegestaan

Casting:

- PtrType naar FloatType explicit casts en vice versa zijn niet toegestaan

Implicit Conversions:

- Conversions waarbij precisie verloren gaat, geven warnings
- Conversions tussen PtrTypes geven warnings