Thomas Dooms
Ward Gauderis
2de bachelor Informatica
Compilers

C Compiler

Verslag 2

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: &&, ||, !
- 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).

Opdracht 4:

Verplicht:

- If/else statements
- Loops: while, for, break, continue
- Scopes

Optioneel:

- De switch (samen case en default) implementeren we niet omwille van de grote hoeveelheid semantische uitzonderingen die erbij komen kijken. C laat zeer veel toe in een switch dat logisch gezien geen betekenis heeft en dat niet eenvoudig representeerbaar is door if statements.

Opdracht 5:

Verplicht:

- Functions: definitions, declarations, calls, semantic checks
- Parameter passing
- Return statements
- Function scopes
- Local/global variables
- Void
- Unreachable code na return, control of break wordt verwijderd

Optioneel:

- De check op return statements in alle paden van de code is niet volledig geïmplementeerd. Er wordt enkel gecheckt op een return statement in de functie omdat we anders een uitgebreidere flow control moeten maken om volledig te kunnen zijn.
- Unused variables worden weg geoptimaliseerd
- ConstExpr conditionals worden geëlimineerd

Opdracht 6:

Verplicht:

- Arrays en array operations

- Include van stdio.h: printf, scanf

Optioneel:

- Multi-dimensional arrays
- Assignment van volledige arrays ondersteunen we niet omdat arrays in C non-assignable zijn. Met pointers gaat dit wel.
- Dynamic arrays hebben we niet geïmplementeerd omwille van de garbage collection die zou moeten worden geïmplementeerd in uitzonderlijke gevallen zoals bv. een dynamic array in een loop waarin een break of return staat.

Extra:

- Dereference en address-of operators gecombineerd met multi-dimensional arrays en pointers
- Literal strings als global char arrays die assignable zijn aan char pointers

Grammatica

De grammatica is in dit deel van het project voornamelijk veranderd op vlak van scopes en statements, naast het toevoegen van types zoals literal strings en arrays en enkele expressies zoals function calls en array subscript. De grammatica maakt nu een onderscheid tussen statements, expressies, declaraties en definities. Expressies en statements kunnen niet in de global scope voorkomen, terwijl functiedefinities enkel in de global scope passen. Variabele- en functiedeclaraties (en constexpr definities van variabelen) kunnen zowel globaal als lokaal gebeuren.

Folding

Alles rond folding en optimalisaties gebeurt via recursieve functies in de AST. Als bv. een binaire plus operator merkt dat beide operands gefold kunnen worden at compile time, wordt de node zelf vervangen door de correcte literal. Constant Propagation is in deze zin de folding van een constante variabele in de AST. Ook ‘dead code’ na breaks en return statements wordt hier weggehaald. Vervolgens worden constexpr if statements en for loops weggefolded of worden ze vervangen door een scope. Door deze op dezelfde manier te folden, kunnen we de AST optimaliseren met behulp van één recursieve functie.

Checks

C is zeer laks qua warnings en errors, dit zorgt ervoor dat het moeilijk is de exacte plekken te vinden waar er een warning of error moet voorkomen. De lijst met randgevallen is daar een goed voorbeeld van. Er zijn ook sommige gevallen waar men een error verwacht maar er geen is (bv. 5 / ‘a’). Een groot deel van de functionaliteit van de AST bestaat daarom uit checks om deze allemaal zo volledig mogelijk te rapporteren.

LLVM

Alle functionaliteit van de compiler wordt nu ondersteund in LLVM IR. De IR wordt gegenereerd door een visitor die de AST afgaat en met behulp van de LLVM runtime libraries in memory de intermediate representatie opstelt. Deze wordt dan omgezet naar leesbare IR code in een ‘.ll’ bestand dat kan worden uitgevoerd door clang/LLVM. We hebben een eigen LLVM pass geschreven die ongebruikte code in een basic block zal verwijderen. Indien de optie wordt meegegeven aan de compiler, zullen er ook andere LLVM optimalisatie passes worden uitgevoerd.

Error Handling

Errors en warnings bevinden zich in onze compiler altijd op een bepaald niveau: syntax, semantic, of internal. Syntax errors/warnings worden voornamelijk opgemerkt tijdens het parsen met Antlr, semantic errors /warnings worden ontdekt in de AST en internal errors/warning duiden aan dat er ergens een fout in de compiler is geslopen. 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

Declarations:

- Redeclaration globale variabele versus in andere scope
- Function parameters arrays worden omgezet naar pointers
- Globale variabelen mogen alleen geïnitialiseerd worden door constexpr initializers

Folding:

- Const variabelen waarvan het adres wordt genomen (prefix & operator) mogen niet weggefold worden

Assignment:

- Arrays zijn non-assignable in C