Gymnasiearbete (100 p)
IT-Gymnasiet Göteborg
HT 2017 - VT 2018

Supervisors:
Daniel Berg, David Lundholm

Erwall, a High Level General Purpose Programming Language

30 januari 2018

Created with LaTeX

Author:
Erik Wallström

Abstract

Keywords: Erwall, Programming, Language, C, Compiler

Innehåll

1 Inledning **1**
- 1.1 Bakgrund . . . 1
- 1.2 Syfte . . . 1
- 1.3 Frågeställningar . . . 1
- 1.4 Kravspecifikation . . . 2
- 1.5 Metod och material . . . 3
- 1.6 Teoretisk bakgrund . . . 3

2 Resultatredovisning **3**
- 2.1 Arbetsgången . . . 3
- 2.2 Resultatet . . . 6

3 Diskussion och slutsatser **6**

4 Källförteckning **6**

5 Bilagor **6**

1 Inledning

Jag har valt att skapa ett nytt programmeringsspråk som kan användas praktiskt. Språket kommer att vara kompilerat samt ha statisk typning. Målet är att konstruera en bättre version av det redan existerande programmeringsspråket C. Projektet kommer att ha öppen källkod samt en fri upphovsrättslicens. Jag kommer använda det distribuerade versionshanteringssystemet *git* för att hålla ordning på utveckling och ändring av källkod, samt möjliggöra för andra människor att bidra med förbättringar och ändringar. Detta är ett ganska ambitiöst projekt.

Fokus på programmering och utvecklingen av en kompilator, fokus på språk och standard senare.

Programmering har varit min hobby i nästan 5 år, och under den tiden har jag testat ett stort antal av de existerande programmeringsspråken. Dock så har jag ännu inte hittat något språk som jag anser vara perfekt; antingen saknas funktioner, eller så finns det onödiga och överflödiga funktioner som jag ogillar, med mera. Genom att skapa ett eget programmeringsspråk så kan jag anpassa och förbättra vissa delar så att det blir det perfekta programmeringsspråket för mig.

1.1 Bakgrund

Jag har gått kursen Programmering 1, och går just nu i Programmering 2. Jag har tidigare erfarenhet av många programmeringsspråk samt *git*, samt skapat en interpretator för programmeringsspråket *Brainfuck*. Jag har även konstruerat en textredigerare och utvecklingsmiljö för programmeringsspråket *Lua*, där jag bland annat haft förgmarkeringar för syntax. Språket som jag har mest erfarenhet av är C, vilket även är mitt favoritspråk och grunden till mycket inom detta nya språk. Detta gör att jag bedömer att mina kunskaper är tillräckliga.

Detta projekt kommer fungera som en övning för relativt avancerad programmering, structurering och arbete av stort projekt.

1.2 Syfte

Syftet med detta projekt är att i slutändan ha ett fungerande programmeringsspråk som på sikt kan ersätta programmeringsspråket C, som idag fortfarande är ett av de mest använda programmeringsspråket. Dessutom så ska det visa att ett programmeringsspråk inte är något märkvärdigt; många programmerare tänker klagar på programmeringsspråk, men ytterst få försöker förbättra dem eftersom att mentaliteten att man inte ska återuppfinna det som redan finns, samt att man inte ska byta ut det som redan fungerar. Detta leder till att programmeringsspråk får en speciell status, och många tänker inte på att det rör sig om en viss standard som är satt, eller ett program som helt enkelt tolkar och översätter koden du skriver till maskinkod. Många tycker att det är orimligt att skapa ett nytt programmeringsspråk.

1.3 Frågeställningar

Under projektets gång vill jag försöka utforska och svara på dessa frågor som jag hade från början.

- Vilka delar är en kompilator uppbyggd av och varför?

Alla stora program är uppbyggda av olika delar som tillsammans utför en uppgift. Så vilka delar är det som en modern kompilator för ett språk generellt är uppbyggd av?

- Hur bra fungerar C som ett programmeringsspråk för att skapa en kompilator?

  Debatten om vilka språk som passar för vilka uppgifter har hållit på i många år. Språket som jag har mest erfarenhet av än så länge är C, så hur bra passar det till just det här projektet? Vilka fördelar och nackdelar finns det då det är väldigt nära hårdvaran, samt att standard- biblioteket är minimalt.

- Är C ett bra mål att generera kod till?

  Är det en bra idé att översätta koden skriven i det nya språket till C kod istället för till exemepl maskinkod eller använda virtuella maskiner som till exempel *LLVM*? Vilka fördelar och nackdelar kan det ha?

- Vad krävs av ett programmeringsspråk för att det ska anses som bra"?

  Vad är det egentligen som gör att folk gillar och ogillar språk? Var går balansen mellan bra och användbar? Skaparen av programmeringsspråket C++ har bland annat sagt "Det finns två olika typer av språk; språk som folk klagar på, och språk som inte används".

- Hur ska man planera ett projekt för att tillåta enkel felsökning och tillägg av nya funktioner?

  Stora projekt så som detta kommer vara kommer kräva många ändringar och försvåra felsökning och förbättringar. Vilka verktyg och praxis bör man använda och följa för att förenkla detta?

1.4 Kravspecifikation

- Logisk och konsekvent syntax
- Explicita deklarationer
- Högpresterande och nära maskinvaran

Några specifika specifikationer på språket kommer vara följande:

- Rekursiva omfång (?) med funktioner och kommentarer
- Garanterade svansanrop (?)
- Kompatibelt med C
- Strikt typsystem med riktiga konstanter
- Inget odefinerat beteende
- Listor som första klassens medborgare
- Markerade unioner

  med mera.

1.5 Metod och material

Projektet bygger på research utifrån tekniska artiklar och wikipedia mm. Projektet är skrivet i ren gnu-11 C kod med hjälp av ett textredigeringsprogram (vim) i operativsystemet Arch Linux och kompilerat med kompilatorn GCC. Inga externa bibliotek och API:er har använts, utan allt är skrivet från grunden. För versionhantering har git använts, och för debugging har gdb och valgrind använts.

Motivera....

1.6 Teoretisk bakgrund

This subsection's content...

2 Resultatredovisning

2.1 Arbetsgången

Under de första veckorna var det endast planering, research om ämnet, brainstorming av idéer samt design av syntax som gällde.

2017-10-17

Första prototypen av en lexer samt alla grundläggande strukturer blev klara idag. Eftersom att standard-biblioteket i C saknar mycket behövde jag implementera bland annat dynamiska generiska listor, hantering av färgkoder för Linux-terminalen samt en abstraktion av filhanteringgränssnittet. Lexern har enkel felrapportering där både rad och kolumn där felet hittades visas, operatorer, nyckelord, identifierare samt sträng-, boolean- och nummerlitteraler, mångradiga kommentarer som kan vara nested.

2017-10-19

Idag blev första parser-prototypen färdig. Jag skrev ett gränssnitt för abstrakta syntaxträd, och lade till logisk utmatning så att man enkelt och grafiskt kan se hur trädet är uppbyggt. Parsern är en recursive descent parser, som använder tokens från lexern samt AST-gränsittet för att konstruera ett träd. Just nu stödjer det bland annat hantering av funktioner och block, med logiska felrapporter. Jag upptäckte att syntaxen för flerradiga kommentarer orsakade problem i vissa specifika fall, och ändrade därför i lexern för att lösa detta.

2017-10-20

Parsern stödjer nu numeriska uttryck med alla vanliga matematiska operatorerna, funktionsanrop samt variabeldeklarationer. Lexern stödjer nu även bitwise-operatorer.

2017-10-22

Nu stödjer parsern även if-satser och booleanska uttryck.

2017-10-23

Idag började jag arbeta på semantisk analys av AST:t. Kompilatorn rapporterar nu om odefinerade variabler, funktioner och typer, samt när du anropar en funktion med inkorrekt antal argument med mera. Jag skapade ett även ett gränssnitt för scopes, fixade så att parsern nu kan hantera typdeklarationer och return statements, och började komma på idéer om syntax för listor samt referenser för språket.

2017-10-28

Jag har nu lyckats lägga till tester för logiska och numeriska operationer och funktionsdeklarationer. Parsern stödjer nu även typomvandlingar.

2017-10-29

Idag lade jag till så att parsern klarar av elseif och else satser, samt externa funktionsanrop. Dessutom lade jag till semantiska kontroller för main-funktionen.

2017-11-01

Nu har jag lagt till experimentell kodgenerering. Jag var tvungen att själv skapa en abstraktion av C-stränggränssnittet eftersom att det inte var tillräckligt användbart för mina ändamål.

2017-11-02

Kompilatorn stödjer nu argument för att stödja frivillig utskrivning av AST och tokens med mera. C har inget sådant bibliotek i standarden, så jag fick skriva en API för POSIX-baserade kommandoradsargument.

2017-11-03

Kompilatorn stödjer nu automatisk kompilering av den genererade C koden med hjälp av GCC.

2017-11-04

Parsern och kodgeneratorn stödjer nu tomma typer. Generatorn skapar nu korrekt indenterad C kod, och genererar if-satser korrekt. Semantiska tester för externa funktionsanrop samt memory leaks i semantiska tester är nu fixade.

2017-11-05

Små förbättringar av funktionsgenereringen lades till.

2017-11-07

Idag lade jag till generering av uttryck samt extra semantisk information in i AST noder-

na. Experimentell generering av lokala funktioner och funktionsanrop lades även till.

2017-11-08

Idag fixade jag en bug som gjorde att lokala variabler inte genererades korrekt.

2017-11-26

Jag har nu arbetat i mer än två veckor med att omstrukturera hela projektet. Anledningen var att felrapporteringen inte var tillräckligt bra, samt att mycket blev rörigt. Det var bättre att strukurera om helt än att ändra på allt som redan var skrivet bedömde jag. Hela projektet är nu betydligt bättre strukturerat vilket gör att ändringar blir enklare. Felrapporteringen är nu tydlig och färgkodad. Semantiska analyser för oanvända och oinitierade variabler fungerar nu korrekt, och parsern stödjer nu tomma return-statements.

2017-12-01

Semantisk analys för att se om funktioner returnerar korrekt fungerar nu. Jag fixade även lite memory leaks som jag upptäckte. Jag började även med enkel optimering, men det visade sig vara betydligt svårare än jag trodde.

2017-12-21

Under veckan har jag testat att implementera en enkelt interpretator för compile-time execution av kod. Efter ett tag bedömde jag dock att detta var ett för stort ämne att börja på nu.

2017-12-29

Jag har lagt till små förbättringar till generator och testerna. Bland annat klarar kodgeneratorn nu externa funktionsanrop, if-satser samt typkonversioner.

2018-01-01

Återigen har jag fixad kodgeneratorn då den fortfarande hade vissa problem. Lokala funktioner samt exponenter i uttryck fungerar nu som de ska göra.

2018-01-05

Idag gjorde jag så att parsern och generatorn hanterar defer-statements.

2018-01-06

While-loopar parse:as och generaras nu korrekt. Jag lade också till förbättringar till felmeddelanden.

2018-01-20

Prestanda-tester har nu lagds till, så att man enkelt kan se hur lång tid varje komponent i kompilatorn tar. Fixade även några små fel som jag hittade.

2018-01-21

Ett riktigt typsystem har nu blivit implementerat. Det kan hantera listor, pekare, funktioner, unioner, enumerationer structurer samt typdefinitioner.

2.2 Resultatet

Kompilatorn och språket är långt ifrån klart. Många av specifikationerna som sattes i början av projektet han inte bli färdiga. Dock så är Erwall nu ett fungerande programmeringsspråk som jag bland annat använde en matematiklektion, vilket tyder på att det redan är användbart och på vissa sätt till och med bättre än C. Bland annat är typsystemet betydligt mer avancerat, och konstanter samt sematiska kontroller och felmeddelande är betydligt bättre i jämförelse.

3 Diskussion och slutsatser

This section's content...

4 Källförteckning

This section's content...

5 Bilagor

This section's content...