CREAREA UNUI COMPILATOR ÎN RUST

Daniel GHENGHEA,

student, anul III, Facultatea de Matematică și Informatică

CZU: 004.42/.43 daniel.ghenghea2000@gmail.com

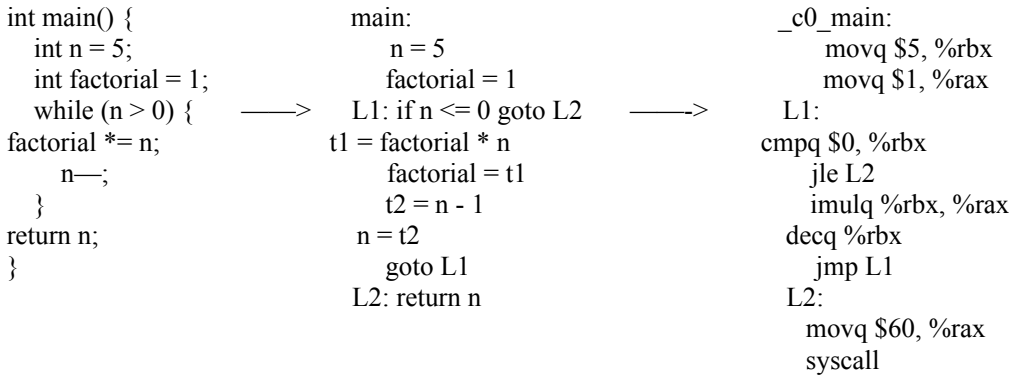
Compilatoarele joacă un rol esențial în traducerea codului sursă dintr-un limbaj de programare într-un altul, facilitând execuția pe platforme hardware diverse. Acest raport își propune să prezinte procesul de dezvoltare a unui compilator pentru C0, un subset al limbajului de programare C, folosind Rust ca limbaj de implementare datorită caracteristicilor sale unice în ceea ce privește siguranța memoriei și performanța.

Alegerea Rust pentru dezvoltarea acestui compilator a fost motivată de câteva caracteristici cheie ale limbajului. Siguranța memoriei, asigurată prin sistemele de ownership și borrowing, elimină erori comune legate de managementul memoriei fără a necesita un garbage collector. Sistemul de tipuri puternic și expresiv al Rust contribuie la prevenirea erorilor la timpul de compilare, în timp ce performanța comparabilă cu cea a limbajelor C și C++ este esențială pentru un proces eficient de compilare.

Dezvoltarea compilatorului a implicat mai multe etape cheie, fiecare abordând diferite aspecte ale procesului de compilare:

1. **Analiza Lexicală și Sintactică:** A început cu transformarea codului sursă într-o listă de cuvinte și apoi într-un Arbore Sintactic Abstract (AST). Acest proces a utilizat un analizator lexical manual și o combinație între Recursive Descent și Algoritmul lui Pratt pentru analiza sintactică, evitând problemele legate de recursia din stânga și stabilind precedența operatorilor într-un mod mai ușor.
2. **Analiza Semantică:** S-a concentrat pe verificarea semanticii programului C0, inclusiv compatibilitatea tipurilor de date, scopul și durata de viață a variabilelor, asigurându-se că funcțiile și variabilele sunt declarate înainte de utilizare.
3. **Reprezentare Intermediară:** S-a folosit o reprezentare bazată pe trei adrese pentru a oferi o formă de abstractizare între codul sursă și codul mașină, facilitând optimizările și adaptarea la diferite arhitecturi hardware. O altă reprezentare analizată este una bazată pe arbori RI.
4. **Generarea Codului:** Ultima etapă a constat în transformarea reprezentării intermediare în cod mașină, cu o atenție deosebită acordată alocării registrelor și selecției instrucțiunilor specifice arhitecturii țintă.

Compilatorul este capabil să transforme un program în limbajul C0 în limbajul de asamblare x86-64, varianta AT&T. Astfel, pornind de la un program ce calculează factorialul unui număr în C0 este transformat după cum urmează:



Implementarea compilatorului pentru C0 în Rust a demonstrat eficiența limbajului în construcția de software de sistem complex. Siguranța memoriei și sistemul de tipuri al Rust au redus considerabil complexitatea și posibilitatea de erori. Se recomandă utilizarea extensivă a pattern matching-ului pentru manipularea AST și adoptarea testării timpurii în dezvoltarea compilatorului pentru a asigura corectitudinea și stabilitatea.

Referințe:

1. 15-411 Compiler Design, Carnegie Mellon University, În:https://www.cs.cmu.edu/~janh/courses/411/24/
2. Engineering A Compiler, 2nd Edition, Keith D. Cooper, Linda Torczon
3. The Rust Programming Language, În:https://rust-book.cs.brown.edu/

Recomandat
***Viorel GRIGORCEA**, lector superior*