Konstrukcija kompilatora

— Osnovno o kompajlerima —

Milena Vujošević Janičić

Matematički fakultet, Univerzitet u Beogradu

Sadržaj

1 Nastanak i namena programskih prevodioca

1.1 Veza kompajlera i programskih jezika

Šta je programski prevodilac?

- Programski jezici su spona u komunikaciji čoveka i računara
- Ljudi i mašine pričaju različitim jezicima, prevodioci se nalaze između
- **Programski prevodilac je program za prevođenje programa iz jezika visokog nivoa u jezik hardvera**.
- Postoji **veliki broj različitih programskih jezika** i za svaki je potrebno da postoji odgovarajući prevodilac
- Postoji **veliki broj različitih mašina** i za svaku je potrebno da postoji odgovarajući prevodilac

Kompajleri

- Već pedesetih godina, u troškovima informatičkih rešenja veći udeo je imao softver nego hardver
- Kompajleri daju mogućnost korišćenja istog koda napisanog na višem programskom jeziku na različitim procesorima

- Prvi komercijalni kompajler (sa karakteristikama današnjih kompajlera) nastaje za prvi viši programski jezik, tj za Fortran (1957)
- Budući kompajleri pratiće osnovne principe Fortrana

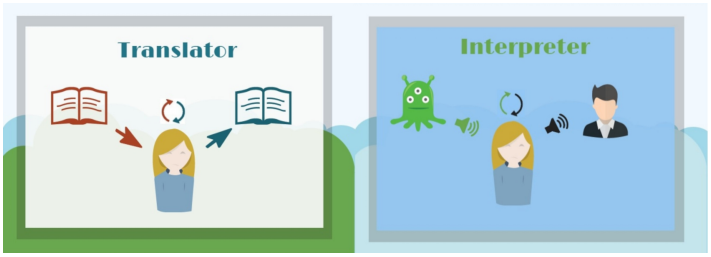
Istorijski podaci

- **Bitne ideje**: In his PhD dissertation (in Mathematics, at ETH Zurich, 1951; published in 1954), Böhm describes for the first time a full meta-circular compiler, that is a translation mechanism of a programming language, written in that same language.
- **Prva implementacija**: (1951–1952) The first implemented compiler was written by **Grace Hopper**, who also coined the term "compiler", referring to her A-0 system which functioned as a loader or linker, not the modern notion of a compiler. https://lemelson.mit.edu/resources/grace-hopper
- **Implementacija prototipa u današenjem smislu kompajlera**: The first compiler in the modern sense were developed by Alick Glennie in 1952 at the University of Manchester for the Mark 1 computer.
- **Implementacija prvog komercijalnog kompajlera**: The FORTRAN team led by John W. Backus at IBM introduced the first commercially available compiler, in 1957, which took 18 person-years to create.

Pristupi implementaciji programskih jezika

- Glavni pristupi implementaciji programskih jezika: **kompajleri** (kompilatori, programski prevodioci) i **interpretatori**

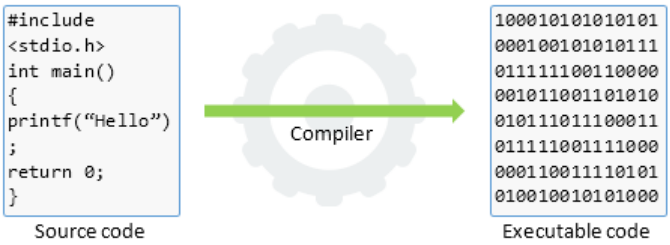
  U kontekstu prirodnih jezika: prevodilac vs simultani prevodilac



Kompilatori (kompajleri)

Kompajler iz jednog programa pravi drugi program, ciljni kod. Ciljni kod može biti asembler, objektni kod, mašinski kod, bajtkod ...

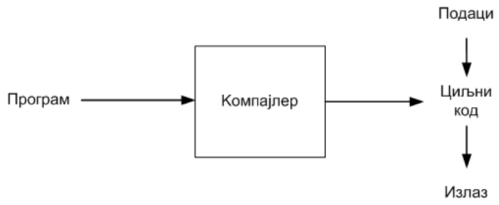
Vrše se optimizacije i kompilator pravi efikasan izvršivi kod.



Nakon prevođenja, dobija se izvršni fajl koji se puno puta može pokrenuti. Originalni program nije više potreban.

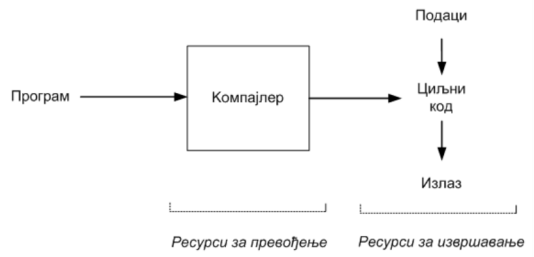
Kompajleri

- Izvršni program se nezavisno pokreće nad podacima, kako bi se dobio izlaz
- Izvršni program se može pokretati željeni broj puta. Pri obradi podataka, kompajler ne učestvuje



Kompajleri

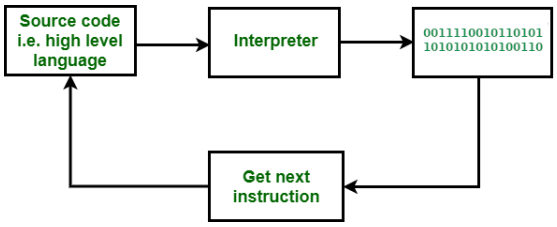
- Resursi za rad programa se dele, tj vremenski i memorijski resursi su raspodeljeni pa samim tim i izvršavanje programa biva brže



Interpretatori

Prevode naredbu po naredbu programa.

Vrše se optimizacije samo na nivou svake pojedinačne naredbe.



U fazi izvršavanja, uvek je neophodno da imamo kod. Prevođenje se svaki put vrši iz početka.

Interpretatori

- Interpreteri ne procesiraju program pre izvršavanja (karakteriše ih sporije izvršavanje, koriste se često za jezike skript paradigme)

Kompilatori vs interpretatori

Prednosti/Mane kompilatora

- Detekcija grešaka unapred
- Kvalitetniji kod (efikasniji, manje zauzeće memorije)
- Vreme prevođenja
- Svaka izmena u kodu zahteva celokupno novo prevođenje
- Svaka izmena arhitekture računara zahteva novo prevođenje

Prednosti/Mane interpretatora

- Detekcija grešaka kada se dese, ako se dese
- Sporije izvršavanje
- Fleksibilnost
- Izmene u kodu su odmah dostupne
- Prenosivost

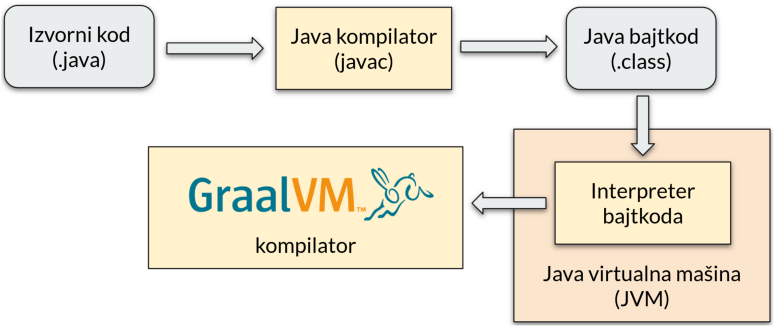
Kombinacija kompilatora i interpretatora

Na koji način je moguće kombinovati ova dva pristupa?

Programski jezik Java

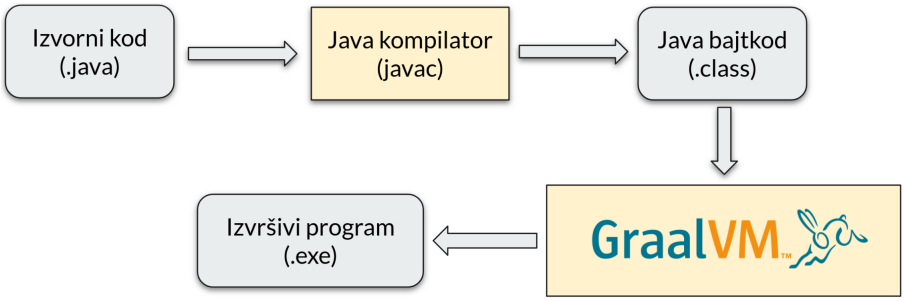
Kombinacija kompilatora i interpretatora

Na koji način je moguće kombinovati ova dva pristupa?
JIT kompilacija — kompilacija u trenutku



AOT Kompilacija

AOT - Ahead of Time



Najpoznatiji prevodioci otvorenog koda

https://gcc.gnu.org/

https://llvm.org/

https://www.graalvm.org/

1.2 Izazovi u razvoju kompajlera

Izazovi programskih programskih prevodioca

Kompajleri su izuzetno kompleksan softver Kompajler treba da...

- ... omogući da je pisanje programa jednostavno
- ... omogući da se izbegnu greške i koriste apstrakcije
- ... prati način na koji programeri razmišljaju
- ... ostvari prostor za što bolji izvorni kod
- ... napravi što bolji izvršni kod: da kreira brz, kompaktan, energetski efikasan kod niskog nivoa

Šta treba da zadovolji kompajler?

Kompajler treba da...

- ... prepoznaje i korektno prevodi samo jezički ispravne programe, to jest da pronađe greške u neispravnim programima
- ... uvek ispravno završi svoj rad, bez obzira na vrstu i broj grešaka u izvornom programu
- ... bude efikasan (posebno da bude brz)
- ... generiše kratak i efikasan objektni program
- ... generiše **odgovarajući** program (semantički ekvivalentan izvornom kodu)

Ispravnost kompajlera

- Ispravnost kompajlera se definiše kao ponašanje kompajlera u skladu sa specifikacijom jezika za koji se vrši prevođenje
- Ispravnost kompajlera je od suštinske važnosti: neispravan kompajler proizvodi neispravne programe (i kao takav je beskorisan, na primer, kompajler koji bi naredbu $a = b + c$ prevodio u naredbu programa koja radi oduzimanje)
- Razvoj kompajlera obuhvata posebne mere za brigu o kvalitetu i o ispravnosti softvera

Ispravnost kompajlera

- Tehnike za razvoja kompajlera obuhvataju formalne metode i rigorozno testiranje
- Za kompajlere koji se koriste za razvoj bezbednosno kritičnog softvera može da se zahteva da budu i formalno verifikovani (najviši nivo ispravnosti softvera: COMPCERT `http://compcert.inria.fr/`)

Dodatni izazovi u razvoju kompajlera

- Omogućiti da specifikacija izvornog programa bude još bliža ljudima, njihovim govornim jezicima i iskustvu: kompajleri deklarativnih jezika, pored leksičke, sintaknse i semantične analize, imaju izražen deo netrivijalne sinteze koda
- Kompajleri za kvantne računare

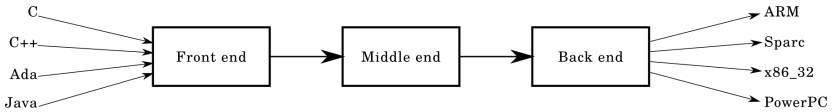
  Programming languages and compiler design for realistic quantum hardware

2 Struktura kompajlera

2.1 Prednji deo (front end)

Prednji deo kmpajlera (front end)

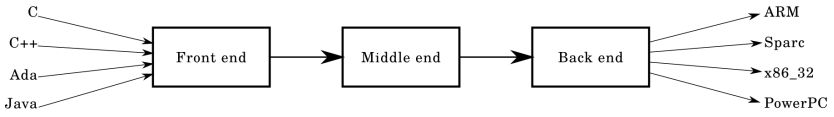
- Prednji deo kmpajlera (front end) zavisi od jezika: vrši leksičku, sintaksnu i semantičku analizu
- Prednji deo kompajlera transformiše ulazni program u međureprezentaciju (engl. *intermediate representation* (IR)) koja se koristi u srednjem delu kompajlera. Ova međureprezentacija je obično reprezentacija nižeg nivoa programa u odnosu na izvorni kod.



2.2 Srednji deo (middle end)

Srednji deo kompajlera (middle end)

- Srednji deo kompajlera (middle end) je **nezavisan od jezika i ciljne arhitekture** i vrši optimizacije
- Srednji deo kompajlera vrši optimizacije na međureprezentaciji koda sa ciljem da unapredi performanse i kvalitet mašinskog koda koji će kompajler proizvesti. Srednji deo kompajlera izvršava optimizacije na međureprezentaciji koje su nezavisne od CPU arhitekture za koju je krajnji kod namenjen.



Srednji deo kompajlera (middle end)

- Srednji deo kompajlera, da bi izvršio kvalitetnu optimizaciju, najpre vrši analizu koda
- **Analiza** obuhvata prikupljanje informacija o programu na osnovu generisane međureprezentacije.
- Postoje razne vrste analiza: analiza toka podataka, analiza zavisnosti, analiza aliasa, analiza pointera...
- Precizne analize su osnova za svaku kvalitetnu optimizaciju. U fazi analize koda, obično se grade i graf kontrole toka i graf poziva funkcija

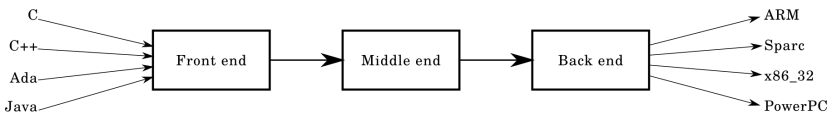
Srednji deo kompajlera (middle end)

- Koristeći rezultate analize, vrši se odgovarajuća optimizacija
- **Optimizacija:** generisana međureprezentacija se transformiše u funkcionalno ekvivalentan ali brži ili manji oblik.
- Popularne optimizacije uključuju *inline*, eliminaciju mrtvog koda, propagiranje konstanti, transformaciju petlji, čak i neko vrste automatske paralelizacije
- Izbor optimizacija koje će biti izvršene zavisi od želja korisnika i argumenata koji se zadaju prilikom pokretanja kompajlera
- Podrazumevan nivo optimizacija obuhvata optimizacije nivoa `-O2`

2.3 Zadnji deo (back end)

Zadnji deo kompajlera (back end)

- Zadnji deo kompajlera (back end) zavisi od ciljne arhitekture i vrši generisanje koda.
- Zadnji deo kompajlera je takođe odgovoran za optimizacije koje su arhitekturalno specifične



Zadnji deo kompajlera (back end)

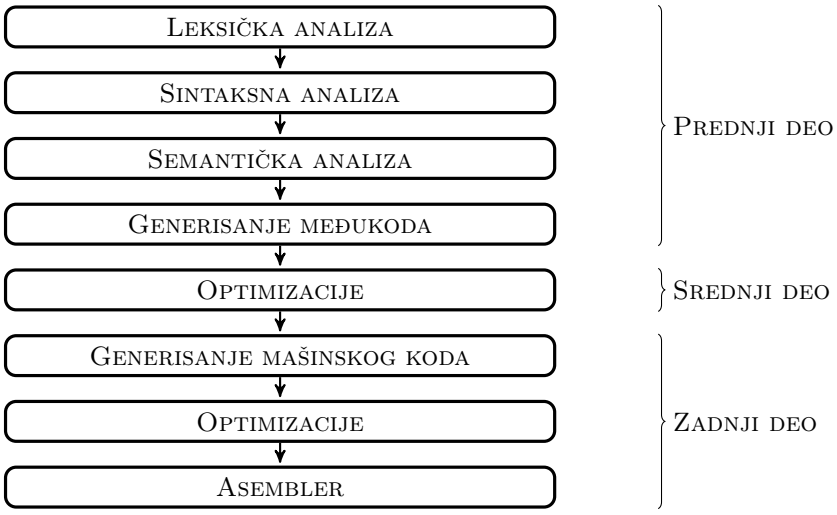
Osnovne faze zadnjeg dela kompajlera obuhvataju **arhitekturalno specifičnu optimizaciju** i genereisanje koda:

- **Mašinski zavisne optimizacije:** optimizacije koje zavise od detalja arhitekture procesora na kojem će se program izvršavati. Ove optimizacije obuhvataju, na primer prepisivanje kraćih nizova instrukcije u odovarajuće instrukcije koje su efikasnije

Zadnji deo kompajlera (back end)

Osnovne faze zadnjeg dela kompajlera obuhvataju arhitekturalno specifičnu optimizaciju i **genereisanje koda**:

- **Generisanje koda:** transformisan međukod se translira u izlazni jezik, obično mašinski jezik sistema. Ovo uključuje odluke vezane za resurse i skladištenje podataka, kao na primer koje promenljive će biti u registrima a koje u memoriji, i izbor i određivanje redosleda instrukcija zajedno sa odgovarajućim tipovima adresiranja. Takođe, podaci za debagovanje treba da se generišu kako bi omogućili debagovnje.



Evolucija kompajlera

- Semantički analizator jača
- Optimizator postaje dominantan