PROGRAMSKI JEZICI i PREVODIOCI

Tekst predavanja

Pripremio: Samir Ribić

Sarajevo, 2016. godine

1. Osnove

Programski jezici su svakodnevnica života softverskog inženjera. Rijetki su programeri koji cijeli svoj radni vijek provedu koristeći samo jedan programski jezik. Programski jezici se mijenjaju nekad iz razloga mode, nekad zbog različitih potreba u softverskom razvoju. Neki programski jezici su pogodniji za određene domene softvera. U mnogim projektima je potrebno kombinovati i više različitih jezika, na primjer C# i PL/SQL, ili C i asemblerski jezik.

Razlozi učenja programskih jezika

Iako većina programskih jezika ima slične osnovne koncepte (postojanje promjenjivih, potprograma, uslova, petlji), kroz komparativnu analizu programskih jezika povećava se mogućnost adekvatnog izražavanja u razvoju aplikacija. Na primjer, pointeri u jeziku C su praktično rješenje za realizaciju drajvera za periferijske uređaje, jer olakšavaju direktni pristup hardveru, dok se operacije s kompleksnim brojevima moraju realizovati bibliotekom. S druge strane, klasični Fortran ima kompleksne brojeve kao prosti tip, ali direktni pristup hardveru zahtijeva eksterne potprograme, pisane npr. u asemblerskom jeziku. Znajući ovaj podatak, čak i bez potpunog poznavanja programskih jezika, postaje jasno u kojem jeziku bi u izboru između ova dva, bi trebalo pisati sistemske, a u kojem programe za numeričke proračune. Neki jezici imaju i radikalno drugačije koncepte. Funkcionalni i logički programski jezici se orijentišu na pitanje “Šta uraditi”, više nego “Kako uraditi” i prepuštaju proces zaključivanja računaru.

Upoznavanje više programskih jezika kroz komparativnu analizu olakšava kasnije upoznavanje novih. Većina modernih programskih jezika sintaksno podsjeća na C (vitičaste zagrade, for ,if, while, switch), pa je nakon C-a lako preći na C++, Java, PHP, JavaScript ili Perl, dok linijski koncept BASIC-a vodi ka lakšem prihvatanju Fortran-a ili Python-a.

Kroz poređenje s drugim jezicima, postaju jasniji i neki koncepti u već poznatim programskim jezicima. Zašto Java, iako objektno orijentisani jezik zamišljen da bude jezik visokog nivoa ima tako kriptičnu sintaksu? Odgovor je istorijski: zbog uticaja jezika C.

Primjene

Različiti programski jezici su nastajali u cilju pokrivanja različitih oblasti razvoja softvera. Računari su se prvo koristili u vojsci i naučnim institutima za rješavanje **numeričkih problema**. Ovakve aplikacije su zahtijevale uvođenje računa u pokretnom zarezu uz upotrebu nizova i potprograma i prvi jezik koji je omogućio njihov efikasan razvoj je Fortran**. Poslovne aplikacije** se koncentrišu na pravljenje izvještaja,. Kod ovakvih aplikacija ne koristi se pokretni zarez, jer finansijski izvještaji nemaju smisla sa vrijednostima manjim od jednog centa, nego su uvedeni decimalni brojevi s fiksnom tačkom i rad s znakovima. Prvi predstavnik jezika koji zadovoljavaju ove zahtjeve je COBOL, a u novije vrijeme PL/SQL . Oblast **vještačke inteligencije** se uglavnom bavi manipulacijom simbolima a ne brojevima. Rani jezi koji simbole drži kroz ulančane liste je LISP. Za **predstavljanje algoritama** programski jezik treba da stimuliše čitljiv kod, pa su nastali Algol i Pascal. **Sistemsko**

programiranje zahtijeva efikasnost zbog stalne upotrebe i tu se pokazao uspješnim C. **Veliki projekti** doveli su do potrebe ponovne upotrebljivosti programa, što je olakšano objektnim jezicima poput C++ i Java. **Web Softver** se bazira na kolekcija jezika za označavanje (HTML), serversko skriptiranje (PHP), klijentsko skriptiranje (JavaScript). Jezici **opšte namjene** nisu strogo vezani za određenu oblast primjene, nakon manje uspješnih PL/1 i Ada, primjenu su stekli C++, Java, Delphi i drugi.

Kriteriji ocjene jezika

Pri ocjeni programskih jezika, koristiće se sljedeći kriteriji. Čitljivost, Mogućnost pisanja, Pouzdanost: je usaglašenost s svojim specifikacijama. Cijena je ukupna cijena upotrebe programskog jezika

Čitljivost

Čitljivost je lakoća kojom se jezik čita i razumije. Ima puno parametara koji utiču na čitljivost.

Jednostavnost jezika je jedan od parametara koji utiču na čitljivost. Jednostavnost se postiže pravilnim **izborom skupa mogućnosti i konstrukcija.** Ako jezik ima previše mogućnosti (C++) a čitalac i pisac aplikacije poznaju različite podskupove jezika, program će biti težak za čitanje. Slično, ako je mogućnosti premalo, (asemblerski jezik) program će opet biti slabo čitljiv jer se konstrukcije moraju realizovat s velikim brojem naredbi. **Dupliciranа funkcionalnost** negativno utiče na jednostavnost. Pascal, npr, ima samo jedan način za uvećanje varijable za 1 (a:=a+1), dok u jeziku C ih ima čak četiri. **Preopterećenje operatora** takođe negativno utiče na jednostavnost, npr. znak + je logičan simbol za sabiranje brojeva i spajanje stringova, ali prilično loš izbor operatora za množenje kompleksnih brojeva.

Ortogonalnost je sljedeći od parametara koji utiču na čitljivost. Jezici imaju mali skup primitivnih konstrukcija koje se kombinuju na mali broj načina. Jezik je savršeno ortogonalan ako je svaka moguća kombinacija legalna i predvidljiva. Primjeri koji narušavaju ortogonalnost su nemogućnost funkcija u Pascal-u da vrate vrijednost koja nije prosti tip, ili različito ponašanje a=a+2 naredbe u C-u zavisno od toga da li je varijabla cijeli broj ili pointer. Forth je vjerovatno najortogonalniji jezik jer njegovi potprogrami i varijable imaju istu sintaksu, a imena im mogu sadržati bilo koji znak (no ovaj jezik ne spada u čitljive zbog ograničenog broja tipova podataka).

Podržani tipovi podataka su bitan parametar koji doprinosi čitljivosti. Stariji programski jezici imaju samo ugrađene tipove, a kasniji jezici mogu da definišu vlastite. Ako je skup ugrađenih tipova nedovoljan, moraju se koristiti neoptimalni tipovi (npr. C nema logički tip pa se on simulira cijelim brojem, a rani Fortran i BASIC nemaju strukture za čuvanje srodnih podataka u istoj sintaksnoj jedinici).

Sintaksni elementi utiču na čitljivost. Jedan od njih su **forme identifikatora**. Većina jezika ograničava identifikatore na skup slova i cifara, ali preveliko ograničenje otežava čitljivost. Ekstremni primjer su rane verzije BASIC-a koje ograničavaju imena varijabli na jedno slovo. **Specijalne riječi** obično ne mogu da budu varijable. U nekim slučajevima njihovo dodavanje smanjuje jednostavnost, ali povećava čitljivost (npr. jezik Ada umjesto

vitičastih zagrada ili begin/end ima posebno end if i end loop**). Forma i značenje** doprinose čitljivosti ako značenje programa direktno slijedi iz sintakse. Ako značenje zavisi od konteksta, to ne doprinosi čitljivosti.

Mogućnost pisanja

Mogućnost pisanja je lakoća kojom se pišu programi. U nekim slučajevima kriteriji koji doprinose čitljivosti, **jednostavnost i ortogonalnost**, doprinose i mogućnosti pisanja, jer je programer koji piše programski kod često čita i njegove dijelove koji su ranije napisani. Malo konstrukcija, malo primitiva i mali skup pravila za njihovo kombinovanje čine da programer može da piše u novom jeziku nakon znatno manjeg vremena učenja

Podrška apstrakciji je mogućnost definisanja kompleksnih struktura ili operacija na način da se detalji mogu ignorisati. Na primjer u C++ se struktura binarno stablo može realizovati preko proste klase koja ima dva pointera i numeričku vrijednost u svojim poljima, dok u Fortranu 77 za ovu svrhu treba koristiti tri niza. U ovom slučaju C++ je pogodniji.

Izražajnost je predstavljena kroz skup praktičnih načina za opis operacija. APL ima ugrađene operatore za matrične operacije, a Perl operatore za manipulaciju stringovima. Operator ++ u jeziku C, iako smanjuje čitljivost (duplicirana funkcionalnost) povećava mogućnost pisanja, jer se lako kuca.

Pouzdanost

Pouzdanost: je usaglašenost jezika s svojim specifikacijama pod svim uslovima.

Provjera tipova je najvažniji način povećanja pouzdanosti programskog jezika. Funkcija printf u jeziku C ne provjerava tipove parametara, i ako oni nisu usaglašeni sa prvim parametrom koji predstavlja format ispisa može doći do nepredvidivih posljedica. U jeziku Ada se provjeravaju svi tipovi podataka. Pri tome su dimenzije nizova sastavni dio tipa, pa se mnoga prekoračenja indeksa nizova detektuju već u toku kompajliranja, a ostala u toku izvršenja.

Upravljanje izuzecima omogućava hvatanje grešaka pri izvršenju i preduzimanje korektivnih mjera. C++ i Java imaju konstrukcije kojim se mogu detektovati greške pri izvršenju i preusmjeriti program na korisničku rutinu.

Aliasi predstavljaju prisustvo dva ili više različitih metoda referenciranja na istu memorijsku lokaciju. Na primjer u C-u postoji union struktura čiji svi elementi dijele isto područje memorije, dok više pointera može da pokazuje na istu lokaciju. Kako se ovim miješaju tipovi podataka, aliasi smanjuju pouzdanost.

Čitljivost i upisivost obično doprinose pouzdanosti, jer olakšavaju reprezentaciju algoritma. Jezik koji ne podržava “prirodan” način predstavljanja algoritma će zahtijevati “neprirodne” pristupe i time smanjiti pouzdanost

Cijena

Cijena je ukupna cijena upotrebe programskog jezika, mjerena u novcu i vremenu. Na cijenu utiču

- Troškovi obučavanja programera da koriste jezik
- Trošak pisanja programa, koliko je jezik blizak datoj aplikaciji.. Npr, COBOL nije dobar za trigonometrijske proračune
- Brzina kompajliranja programa
- Brzina izvršavanje programa
- Cijena samog kompajlera (neki su i besplatni)
- Pouzdanost, loša pouzdanost vodi većim troškovima
- Troškovi održavanja programa najviše zavise od čitljivosti

Ostali kriteriji ocjenjivanja

Portabilnost je lakoća kojom se programi mogu prebaciti s jedne implementacije na drugu Neki jezici su dobro standardizovani, pa je proces prebacivanja minimalan.

Generalnost je primjenljivost programskih jezika na širok opseg aplikacija koje se mogu u njima razvijati.

Dobra definisanost je kompletnost i preciznost zvanične definicije jezika

Uticaji na dizajn jezika

Dva najvažnija faktora koji su uticali na dizajn programskih jezika su računarska arhitektura i programerske metodologije

Uticaj računarske arhitekture

Dobro poznata računarska arhitektura, Von Neumannova arhitektura je uticala na nastanak imperativnih jezika. Arhitektura je toliko rasprostranjena da su ovi jezici dominantni.

Von Neumanova arhitektura pretpostavlja da su podaci i programi smješteni u memoriji. Memorija odvojena od procesora i podijeljena je u memorijske ćelije.. Podaci i instrukcije se prebacuju iz memorije u procesor a procesor izvršava instrukciju po instrukciju sljedećim algoritmom.

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the counter
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

Arhitektura Von Neuman je vodila ka prvim programskim jezicima visokog nivoa. Instrukcije u ovim jezicima se pišu sekvencijalno i prate kretanje programskog brojača, dok varijable modeliraju memorijske ćelije.

Uticaj programerskih metodologija

U ranim periodima računarstva, 1950-e i rane 1960-e, razvijane su proste aplikacije, uz briga o efikasnosti računara. U kasnim 1960-im,: efikasnost ljudi

postaje važnija. To je zahtijevalo veću čitljivost i bolje kontrolne strukture. Metodologija razvoja “Odozgo prema dolje” uticala je na razvoj jezika koji imaju sređene programske strukture i bolju kontrolu tipova.

Kasne 1970-e: dovode do novog pogleda na metodologiju razvoja softvera. Do tada su programski jezici bili procesno orijentisani. Nakon ovoga, oni su sve više orijentisani ka podacima i toku podataka. i raniji programski jezici su imali apstrakciju programa, to jest sakrivanje nepotrebnih detalja implementacije u cilju pojednostavljenja cjeline programa, kroz potprograme. A sada se uvodi apstrakcija podataka, mogućnost definisanja korisničkih tipova podataka i operacija nad njima u istoj sintaksnoj jedinici, prvi put u jeziku Simula 67. To je dovelo do danas najpopularnije paradigme,. objektno-orijentisano programiranje , koje kombinuje apstrakcija podataka, nasljeđivanje i polimorfizam.

Kompromisi u dizajnu jezika

Poboljšanje jednog kriterija u dizajnu jezika ponekad ide na uštrb drugih. Povećanje pouzdanosti ide na uštrb cijene izvršenja. Primjer: Java zahtijeva da se svi pristupi elementima nizova provjere u opsegu što povećava vrijeme izvršenja. Nekada se bira između čitljivost i upisivosti. Primjer: APL pruža mnogo moćnih operatora i velik broj novih simbola što omogućava da se kompleksan račun obavi u kompaktnom programu po cijenu loše čitljivosti Slično, povećanje mogućnosti pisanja može da smanji pouzdanost, primjer je uvođenje pointera u C koji su najčešći uzroci rušenja programa.

Metode implementacije

Programski jezici se implementiraju u osnovi na tri načina : Kompajliranje (programi se prevode u mašinski jezik), čisto interpretiranje (programi se interpretiraju drugim programom koji se zove interpreter) i hibridni Implementacioni sistemi (kompromis između kompajlera i čistih interpretera)

Kompajliranje

Kompajliranje je prevođenje programa u višem programskom jeziku (izvorni jezik) u mašinski jezik . Osobina ovog pristupa je da prevođenje zahtijeva izvjesno vrijeme, ali izvršavanje je dosta brzo, jer su programi prevedeni u mašinski jezik. Proces kompilacije ima više faza

- Leksička analiza: prevodi znakove izvornog programa u leksičke jedinice
- Sintaksna analiza: prevodi leksičke jedinice u stablo parsiranja koje predstavlja sintaksnu strukturu programa
- Semantička analiza: generiše među-kod
- Generisanje koda: generiše se mašinski kod

Nakon kompletiranja dobija se prevedeni modul On se povezuje sa sistemskim modulima da bi se dobio modul za učitavanje u procesu **povezivanja**.

Čisti interpreter

Kod **interpretera** se dijelovi programa napisani na nekom

programskom jeziku, nakon sintaksne provjere, odmah izvršavaju. Za razliku od kompajlera, ne mora cijeli izvorni program da bude prvo preveden u drugi programski jezik:

Prednost interpretera se sastoji u tome da se pojedini dijelovi programa mogu mijenjati, a da se cijeli program ne mora ponovo prevoditi. Interpreteri omogućavaju lakšu implementaciju programa, jer se greške u izvršenju se mogu lako i odmah pokazati.

Mana interpretera je u sporije izvršenje (10 do 100 puta sporije od kompajliranih programa). Često zahtijevaju više prostora jer je u RAM memoriji pored korisničkog programa potrebno imati i interpreter (mada na računarima iz 80-ih godina interpreter za BASIC je često bio ugrađen u ROM).

Čisti interpreteri se rijetko koriste za tradicionalne programske jezike višeg nivoa (C, Fortran), ali su se vratili na scenu s Web skriptnim jezicima (npr., JavaScript, PHP)

Hibridni Implementacioni sistemi

Kompromis između kompajlera i čistih interpretera postignut je hibridnim implementacioni sistemima. Kod ovih pristupa se programski jezik visokog nivoa se najprije cijeli prevodi u među-jezik, koga je brže analizirati interpreterom nego izvorni kod u datom jeziku. Nakon ovoga taj među-kod se izvršava na jedan od sljedećih načina.

Bytecode izvršavanje, primijenjeno u jezicima Perl i ranijim verzijama jezika Java, izvršavaju programe za virtualnu mašinu koja ima instrukcije kao i stvarni mikroprocesori, ali čiji je skup instrukcija više prilagođen izvornom jeziku i dizajniran za lakšu analizu. Byte code, pruža portabilnost na bilo koju mašinu koja ima byte code interpreter i run-time system (zajedno se zovu Java Virtual Machine)

Just-in-Time Implementacija u prvoj fazi radi jednako. U početku se prevede program u među-jezik. U trenutku učitavanja, među-jezik. se prevede u potprograme mašinskog koda koga poziva. Verzija u mašinskom kodu se čuva u memoriji za kasnija pozivanja, tako da osim vremenske zadrške na startu daljnje izvršavanje dobijamo performanse izvršenja uporedive s kompajliranjem. Ovi sistemi se koriste za Java programe i .NET jezike.

Preprocesori

Preprocesorski makroi (instrukcije) se koriste da se navede da će se uključiti kod iz druge datoteke. Preprocesor obrađuje program neposredno prije kompilacije za ekspanziju makronaredbi. Poznati primjer je C preprocesor koji ekspanduje #include, #define, i slične naredbe

Programska okruženja

Programska okruženja su skup alata za softverski razvoj. Mogu se sastojati od editora, kompajlera i linkera ali i uključivati širog spektar alata dostupan kroz jedinstveni korisničko interfejs. Primjera za okruženja ima više.

U slučaju UNIX-a, uz sam operativni sistem dolazi i kolekcija razvojnih alata. Iako generalno orijentisan ka komandnoj liniji. sada je korišten kroz GUI (npr., CDE, KDE, ili GNOME) iznad UNIX

Microsoft Visual Studio.NET je veliko, kompleksno vizuelno okruženje. Korišten za gradnju Web aplikacija ali i onih koje nisu vezane za njega .NET jezicima

NetBeans i Eclipse su srodni Visual Studio .NET, namijenjeni za aplikacije u Java

Rezime

Studija programski jezika je važna iz sljedećih razloga :povećanje kapaciteta za upotrebu raznih konstrukcija, omogućuje inteligentnih izbor jezika omogućava lakše učenje novih jezika.

Najvažniji kriteriji za ocjenu programskih jezika uključuju čitljivost, upisivost, pouzdanost, cijenu.

Glavni uticaji na dizajn programskog jezika su bili mašinska arhitektura i metodologije softverskog dizajna.

Glavne metode implementacije programskih jezika su kompajliranje, čisto interpretiranje i hibridna implementacija.