Datum:6.10.2015.

PROGRAMIRANJE I PROGRAMSKI JEZICI

UVOD

Još od vremena kada je matematičar John Von Neumann, potaknut uspjehom koncepcije ENIAC računara, otpočeo (1945. godine) apstraktnu studiju obrade podatka na računaru koja je pokazala da je računar sa jednostavnom, fiksnom fizičkom strukturom u mogućnosti da izvrši bilo kakvo izračunavanje ako postoji prava programska kontrola, pa do današnjih dana, jasno je da je programiranje suštinski dio aktivnosti čovjeka u njegovom nastojanju da što efikasnije iskoristi računar kao nezamjenjivi alat u gotovo svim oblastima života i rada.

Čovjek je, u svom razvoju, uvijek bio svjestan značaja informacije kao dragocjenog resursa i nastojao je da, na što brži način, dođe do informacije i da je na najefikasniji način iskoristi u rješavanju problema. Ta nastojanja ostvarena su, u neslućenim razmjerima, upravo zahvaljujući razvoju računara kao tehničkog pomagala kao i metoda njegovog efikasnog korištenja.

Efikasno korištenje računara upravo najviše ovisi o **programu** kao proizvodu programiranja jer predstavlja sučelje između čovjeka i računara. Programiranjem se bavi čovjek nastojeći da u tom procesu stvaranja programa realizira model sistema iz realnog svijeta. Sasvim je logično da taj model kao rezultat ljudske apstrakcije ili slike (dinamičkog procesa) sistema nosi obilježja slike čovjeka o samom sebi i njegove interakcije sa okolinom.

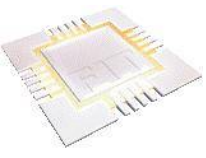
Model kao najjednostavniji pogled na sistem iz realnog svijeta čine objekti sa njihovim svojstvima i međusobnim odnosima. Objekti i odnosi predstavljaju se kroz svojstva koja opisujemo pojmovima. Pojmovi su ustvari koncepti koje koristi čovjek u rješavanju problema.

Podatak je osnovni element informacijske slike svijeta. Podatke koristimo u svim procesima, iz njih spoznajemo objekte i sisteme iz realnog svijeta i na njih, organizirano (planski) djelujemo. Obrađeni podatak daje informaciju kojom spoznajemo zakonitosti realnog svijeta i povećavamo količinu znanja.

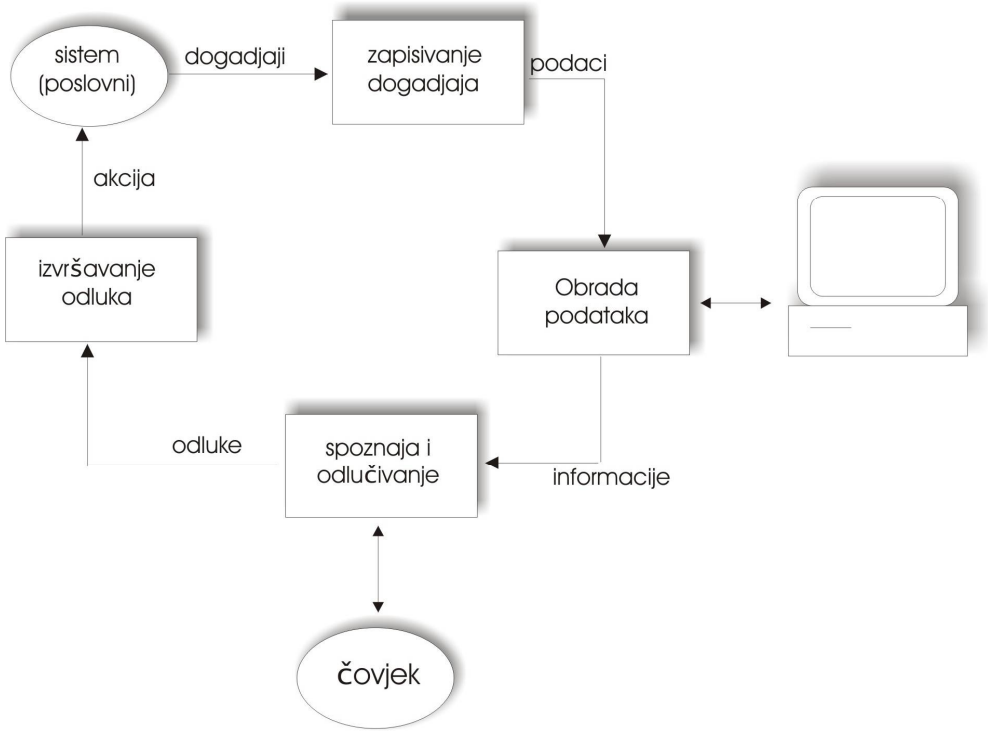
Računar kao alat koji uvećava čovjekove sposobnosti sagledavanja, sposobnosti primanja i obrade podataka i informacija, sposobnosti zaključivanja i donošenja odluka razvijan je tako da omogući prijem, skladištenje, obradu i slanje podataka, informacija i znanja.

Rezultat primjene informacije može se postići samo u zatvorenom informacijskom krugu tj. ako je informacija dostupna subjektu i ako djeluje na određeni sistem, a to se upravo postiže programom.

Nagli razvoj računara i računarskih sistema utiče na to da on preuzima sve više funkcija u informacijskom krugu a čovjeku ostavlja sve više mogućnosti da se aktivira na višem nivou odlučivanja i upravljanja.



Informacijski krug



ŠTA JE PROGRAM

Programiranjem se proizvodi sučelje (interface) za komunikaciju ljudi sa računarima i računara međusobno.
Razlog komunikacije je prijenos "znanja" u vidu podataka i **algoritama** (model koji definira akcije koje treba izvršiti po tačno utvrđenom redoslijedu i u konačnom vremenu), te izdavanje naredbi za obavljanje određenih radnji. U osnovi se komuniciranje ostvaruje primopredajom poruka.
Svaki zadatak koji se zadaje računaru u vidu poruke – programa, sadrži neki algoritam. Prema tome, priprema poruka za računar – što zovemo "programiranje" sastoji se od kreiranja odgovarajućeg algoritma i njegovog zapisivanja u programskom jeziku. Proizvod tog postupka je program.

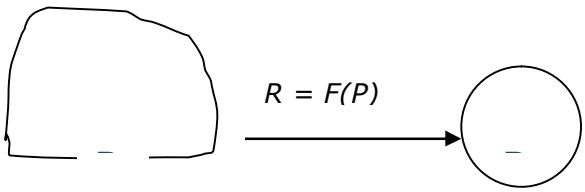
Manje formalnim jezikom program možemo definirati kao niz naredbi, primitivnih i složenih, kojim se opisuje postupak unosa, obrade (izračunavanja) i zapisa podataka i rezultata obrade.

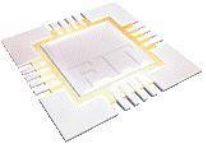
Programiranje je proces zadavanja skupa naredbi u nekom programskom jeziku da bi se izvršila neka aktivnost, odnosno, riješio određeni problem.

Upotreba znanja u rješavanju problema, u suštini, je i osnovna funkcija programa. Korištenje znanja vezano je uz pojam inteligencije. To bi značilo da se program može promatrati kao inteligentni operator ***F*** koji ostvaruje rješenje ***R*** danog problema ***P***.

R = F(P)

Jasno je da ovakva definicija spada u viši nivo apstrakcije jer bi se ovakav stav mogao primijeniti i na osnovnu funkciju informatike: rješavanje problema korištenjem računara.

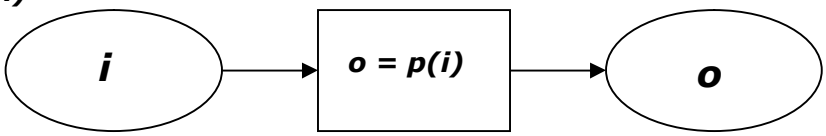




Konkretniji pogled na program (bliži računarstvu) je ako ga promatramo kao funkciju transformacije podataka. Slika problema koji rješavamo opisuje se podacima. Parametri koje zadajemo definirajući ciljeve također su podaci. I jedna i druga kategorija su ulazni podaci, tj. čine skup ulaznih podataka u računar. Prethodno definirana (očekivana) svojstva rješenja problema postaju podaci (pridružuju im se konkretne vrijednosti) u toku izvršavanja programa. To su izlazni podaci računara.

Dakle, program je funkcija ***p*** koja transformira ulazne podatke ***i*** u izlazne podatke ***o*.**

o = p(i)



Programom je omogućeno željeno "ponašanje" računara. Računar, sam po sebi, nema prethodno definiranu informacijsku funkciju niti inteligenciju da bi se direktno mogao iskoristiti prije nego mu se ta svojstva "daju" kroz program.

Programom se računaru daju direktive kojima se definiraju svojstva podataka, postupci sa podacima i pravila po kojima se ti postupci izvode.

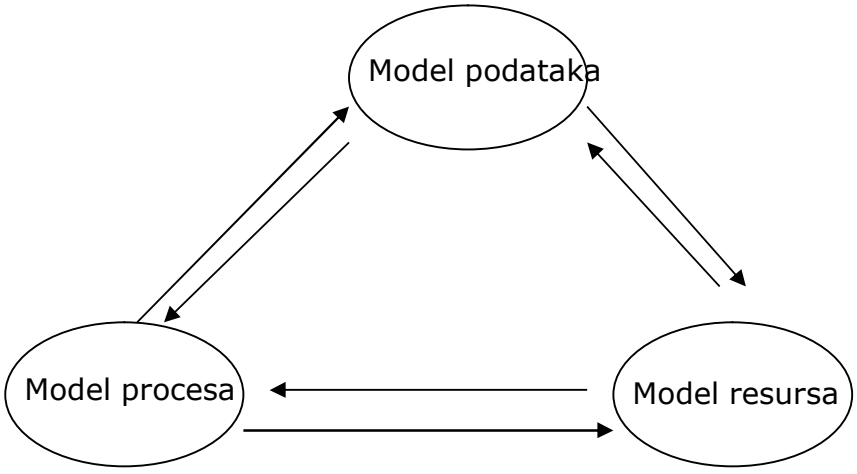
Već smo napomenuli da se rješenje problema najčešće predstavlja modelima.

Uobičajeni način modeliranja rješenja treba da sadrži: model podataka, model procesa i model resursa.

Model podataka sadrži definicije podataka (popise atributa) i strukture podataka (nizovi, polja, datoteke, tabele, liste, stabla itd.).

Model procesa treba da pokaže redoslijed izvršavanja procesa, a način prikazivanja se realizira dijagramima toka, dijagramima strukture, dijagramima akcija i dr.

Model resursa definira hardverske i softverske resurse i njihove najvažnije (relevantne) karakteristike.



Nema zamjene za :

– Inteligenciju

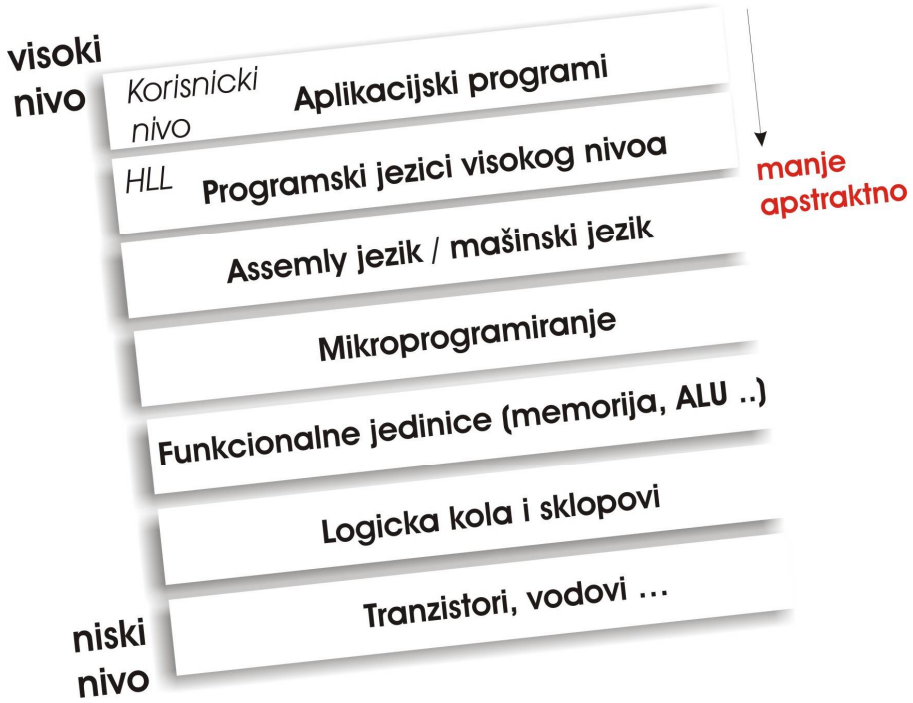
– Iskustvo

– Ukus

– Naporan rad

HIJERARHIJA PROGRAMSKIH JEZIKA

- hijerarhija jezika kroz hijerarhiju kompjuterskog sistema



PREGLED RAZVOJA PROGRAMSKIH JEZIKA

Programski jezici su samo jedan dio ukupnog skupa jezika sa kojima se često susrećemo. Za razliku od "živih", odnosno govornih jezika koji su nastajali, razvijali se pa iščezavali u dugim vremenskim intervalima, kako su se razvijale pojedine grupacije ljudi (narodi), programski jezici "traju" često veoma kratko vrijeme, a samo je mali broj onih koji su se održali u upotrebi nekoliko decenija. Osim toga, programski jezici se, u nastanku, precizno i formalno definiraju i rijetko doživljavaju promjene za razliku od govornih koji su nastali spontano i neprekidno se mijenjaju i razvijaju.
I formalni (umjetno nastali) i prirodni (govorni) jezici imaju mnogo zajedničkog.
Jezik nastaje u cilju ostvarivanja komunikacije među subjektima komuniciranja.

Prvi računari koji su se pojavili bili su vrlo složeni za korištenje. Koristili su ih isključivo stručnjaci koji su bili osposobljeni za komunikaciju s računarom. Komunikacija se sastojala od dva osnovna koraka: davanja uputa računaru i čitanje rezultata obrade. Čitanje rezultata obrade je postalo snošljivo uvođenjem štampača na kojima su se ispisivali rezultati. No, davanje uputa – programiranje – se sastojalo od mukotrpnog unosa nula i jedinica. Pomoću nizova nula i jedinica su se davale upute računaru kao što su: "zbroji dva broja", "premjesti podatak s jedne memorijske lokacije na drugu" i sl. Kako je takve programe bilo vrlo složeno pisati, a još složenije čitati i ispravljati greške u njima, ubrzo su se pojavili prvi programerski alati nazvani asembleri.

U asemblerskom jeziku svaka mašinska instrukcija predstavljena je mnemonikom koji je razumljiv ljudima koji čitaju program. Na primjer, operacija zbrajanja se predstavljala mnemonikom ADD, a premještanje podataka s jedne lokacije na drugu mnemonikom MOV. Upotrebom asemblera se omogućila bolja čitljivost programa, ali je

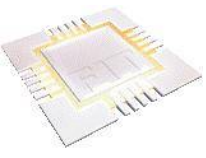
programiranje i dalje bilo mukotrpan posao. Osnovni problem koji je trebalo riješiti, a čije rješenje je dovelo do razvoja mnoštva različitih programskih jezika, je potreba za programskim alatom koji će programere osloboditi rutinskih poslova te mu omogućiti da se fokusira na problem koji rješava. Navedeni problem je potakao razvoj velikog broja programskih jezika, koji omogućavaju programeru da se oslobodi rutinskih poslova. Programski jezik FORTRAN je bio prilagođen matematičkim proračunima, BASIC je bio jednostavan za učiti i kao takav postao najčešće korišten program u obrazovne svrhe, a COBOL je bio namijenjen upravljanju bazama podataka.

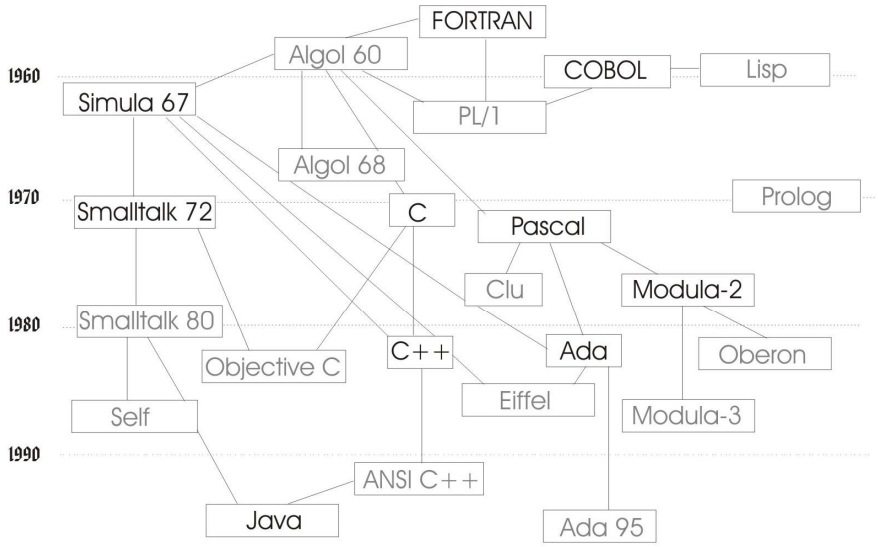
Ranih 70-tih godina 20. stoljeća pojavljuje se programski jezik C, koji je direktna preteča današnjeg C--. Programski jezik C je bio prvi jezik opće namjene i kao takav postao je iznimno popularan. Neki od razloga njegove popularnosti su svakako jednostavnost i modularnost. Osim toga, program je omogućavao dobru kontrolu mašinskih resursa, te je na taj način omogućio programerima da optimiziraju svoj kod.

S razvojem programske podrške došlo je i do promjene u oblasti programskih jezika. Složeni projekti od nekoliko stotina hiljada linija koda postali su uobičajeni, pa je bilo potrebno uvesti mehanizme kojim će se takvi programi učiniti jednostavnijim za izradu i održavanje, kao i mogućnost da se jednom napisani kod koristi u više različitih projekata.

Koncem 70-tih godina 20. stoljeća Bjarne Stroustrup je započeo rad na programskom jeziku "C s razredima". Ovaj programski jezik temeljen je na svojstvima programskog jezika Simula. Iako je programski jezik Simula nudio niz prednosti kao što je pojam razreda (klass) koji je objedinjavao podatke i operacije nad podacima, imao je i mnoštvo nedostataka. Prevođenje programa napisanih u programskom jeziku Simula bilo je dugotrajno, a izvršavanje programa je bilo iznimni sporo. U pokušaju da se riješi nedostataka, a zadrži prednosti Simule i programskog jezika C Stroustrup je započeo rad ne onom što će kasnije postati programski jezik C++. Osim Simule i programskog jezika C, u C++ su ugrađena i svojstva niza drugih programskih jezika kao što su: Clu, Algol68 i Ada.

Do sada razvijeno više od hiljadu programskih jezika, od kojih je samo mali broj prihvaćen od šireg kruga korisnika. C++ je svakako jedan od onih programa koji imaju širi krug korisnika.





Historija razvoja programskih jezika visokog nivoa

OD IZVORNOG KODA DO IZVRŠNOG PROGRAMA

Izvorni kod (sourcve code) je program napisan u bilo kojem programskom jeziku visoke razine. Izvorni kod se u principu može pisati u bilo kojem programu za uređivanje teksta (tekst editor), međutim velika većina današnjih prevoditelja i povezivača (linker) se isporučuje kao cjelina zajedno s ugrađenim programom za upis i ispravljanje izvornog koda. Ovakve programske cjeline poznate su pod imenom IDE – integrated development environment – integrirana razvojna okruženja.

Nakon što se izvorni kod napiše on se pohrani u datoteku izvornog koda na disku. Takve datoteke (napisane u programskom jeziku C++) dobivaju nastavak **.cpp**, .cp ili samo .c.

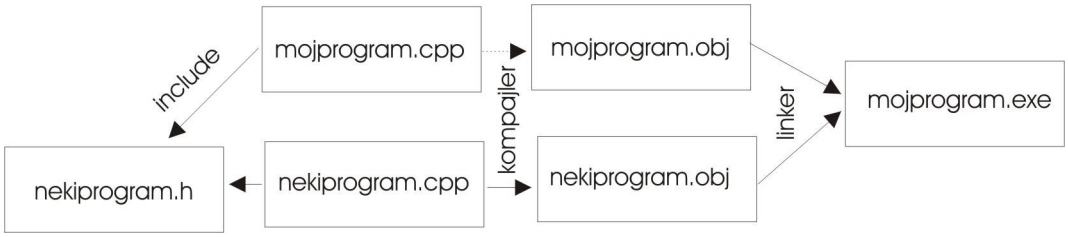
Nakon toga slijedi prevođenje (compile) izvornog koda. U integriranom razvojnom okruženju prevođenje se svodi na pritisak određene kombinacije tipki na tastaturi ili izborom naredbe s nekog od menija.

Prevoditelj (compiler) tokom prevođenja provjerava sintaksu napisanog izvornog koda i u slučaju grešaka ispisuje odgovarajuće poruke. Greške koje se javljaju u ovoj fazi nazivaju se ***compile-time errors*** (greškama pri prevođenju). Sve dok prevođenje izvornog koda ne bude uspješno neće biti moguće pristupiti povezivanju koda.

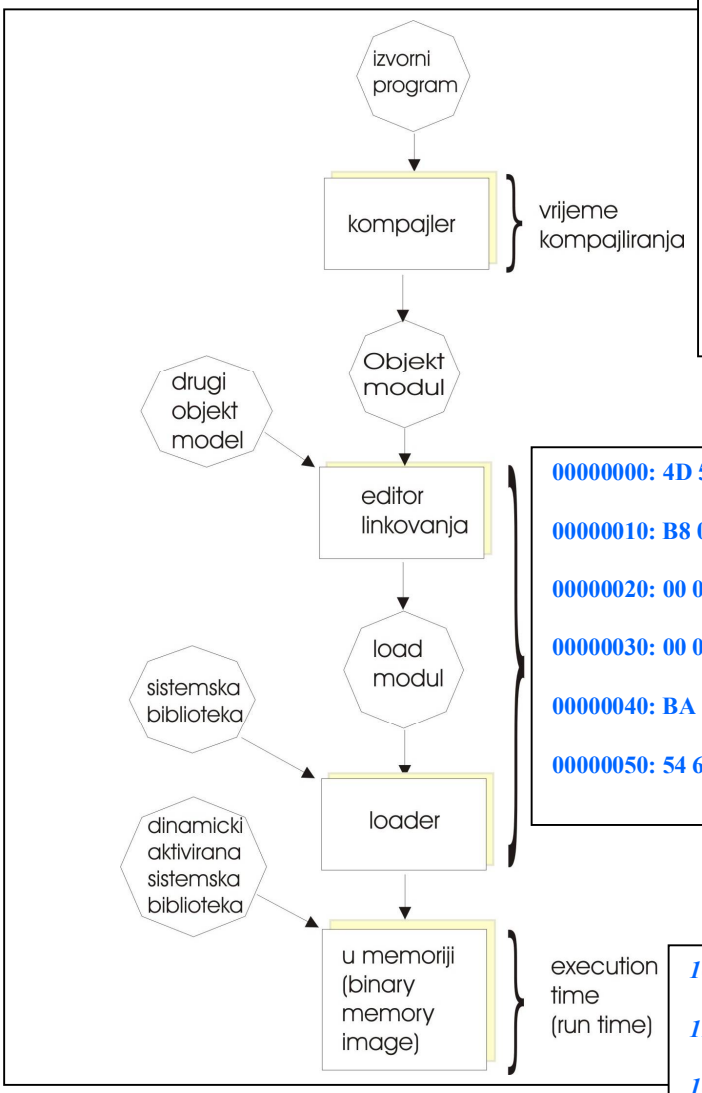
Prevođenjem izvornog koda dobiva se datoteka objektnog koda koju je jednostavno prepoznati po nastavku .o ili **.obj**. U većini slučajeva objektni kod treba povezati s postojećim bibliotekama.

Biblioteke su datoteke u kojima se nalaze već prevedene gotove funkcije i podaci. Biblioteke se obično isporučuju zajedno s prevoditeljem, ali ih programer može i sam kreirati po potrebi. Upotrebom biblioteka se izbjegava ponavljanje pisanje često korištenih operacija. Tipičan primjer za to je biblioteka matematičkih funkcija u kojoj su definirane funkcije poput trigonometrijskih ili eksponencijalnih. Prilikom povezivanja provjerava se mogu li se svi pozivi kodova realizirati u izvedbenom kodu. Uoči li **povezivač** (linker) nepravilnosti prilikom povezivanja, ispisat će poruku o grešci i

onemogućiti generiranje izvršnog koda. Ove se greške nazivaju greškama pri povezivanju – ***link-time errors***. Uspješnim povezivanjem dobiva se **izvršni kod**. No, može se dogoditi da program za jedan skup podataka radi ispravno, a za drugi skup podataka ne. U tom slučaju radi se o greškama pri izvođenju - ***run-time errors***.
Osim pogrešaka prevoditelj i povezivač (linker) redovito dojavljuju i upozorenja. Upozorenja, za razliku od grešaka (errors), ne onemogućavaju nastavak prevođenja, odnosno povezivanja koda, ali predstavljaju potencijalnu opasnost. Cilj je napisati program za koji neće biti niti grešaka, niti upozorenja.




```
#include <iostream>

 float f(float x);

void main()

{

float a;

cin>> a;

cout<<f( a*4);

}
```




```
00000000: 4D 5A 50 00 02 00 00 00 - 04 00 0F 00 FF FF 00 00 MZP.............
00000010: B8 00 00 00 00 00 00 00 - 40 00 00 00 00 00 00 00 ........@.......
00000020: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
00000030: 00 00 00 00 00 00 00 00 - 00 00 00 00 90 00 00 00 ................
00000040: BA 10 00 0E 1F B4 09 CD - 21 B8 01 4C CD 21 90 90 ........!..L.!...
00000050: 54 68 69 73 20 70 72 6F - 67 72 61 6D 20 6D 75 73 This program
```

```
10101101101010110101010101010101110000111011011101111
11011101101110110101101110101101101011001101010101010
10101010101011111100101010101011101010101110101010111
10101010101111110101010101010101011010110101011101111
10101010101011101111101010101010101010101010101010101
10101010101010101010101010101010101010101010101010101
```

VRSTE I OSNOVNE KARAKTERISTIKE PROGRAMSKIH JEZIKA

Po namjeni

- jezici za numeričke i znanstvene probleme
- jezici za poslovne probleme
- jezici orijentirani listama i nizovima
- višenamjenski jezici

Po načinu rada

- imperativni (postižu funkcionalnost postavljanjem vrijednosti varijablama naredbama pridruživanja i ispitivanjem vrijednosti varijabli - FORTRAN, COBOL, ALGOL, C, Pascal, Ada, Modula-2)

- funkcionalni jezici (bez klasičnog pridruživanja, građeni od definicija i poziva funkcija - LISP, ML, LOGO)

- logički ili ciljno orijentirani jezici (postavlja se glavni cilj i daje lista podciljeva čije dostizanje znači i dostizanje glavnog cilja - PROLOG)

- objektno orijentirani jezici (objekti: strukture podataka s definiranim funkcijama nad njima - Smalltalk, Eiffel)

- hibridni jezici (C++)

Generacijska podjela

Jezici prve generacije

Prva generacija: Jezici mašinskog nivoa

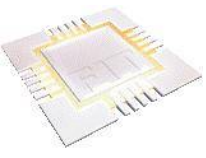
- niski nivo
- pojava s prvim komercijalno raspoloživim računarima (oko 1953. godine)
- programiranje binarnim kodom, tj. zadavanje naredbi i podataka skupinama nula ili jedinica npr. 01010...
- kasnije su dodani mnemonički kodovi instrukcija (CLA 10,..)

Jezici druge generacije - Simbolički jezici (Asembleri)

- srednji nivo
- simboličke umjesto fizičkih adresa memorijskih lokacija (CLA IZNOS)
- asembler = prevodilac mašinsko orijentiranog simboličkog u mašinski jezik
- prevođenje često 1:1
- uvedeni potprogrami
- podjela jezičkih procesora na prevodioce (compiler) i tumače (interpreter)

Jezici treće generacije - Proceduralni programski jezici (3GL)

- jezici visokog nivoa (HLL-High Level Languages)
- jedna naredba jezika prevodi se u više naredbi mašinskog jezika.
- neovisnost o mašini (nije potrebno poznavati arhitekturu, skup instrukcija, registre)
- prenosivost



- danas se može reći da se usavršavanjem različitih programskih jezika dostigao stupanj kod kojeg su uglavnom svi jezici treće generacije višenamjenski.

Jezici četvrte generacije - Četvrta generacija (4GL)

- nagli razvoj tehnologije krajem sedamdesetih i početkom osamdesetih godina
- dostupnost računara povećana potražnja za programskom opremom (software)
- povećanje zahtjeva na SW uzrokovano povećanjem procesne moći računara
- zastarjelost jezika vrlo visokog nivoa (HLL)
- pojava novih alata - jezika
- povećanje složenosti programa i mogućnosti računara
- olakšano korištenje (prirodnija sintaksa, interaktivna sučelja)
- jezici posebne namjene - specijalizirani za određene klase problema
- zadovoljavaju potrebe za izradu 60 - 80 % svih aplikacija

Cilj

- skraćenje vremena izrade
- izbjegavanje kodiranja na niskom nivou
- automatizacija kodiranja (generatori programskog koda)
- lakše testiranje i dokumentiranje
- lakše uvođenje u uporabu i korištenje
- smanjenje troškova održavanja