Složenost i formalizacija algoritama

Složenost algoritama

Prilikom **projektovanja algoritama**, fokus je na **ispravnosti**

Analiza algoritama ima za cilj da pruži **kvantitativnu ocenu njihove efikasnosti**

Potrebno je analizirati kako radi algoritam korak po korak i na osnovu toga zaključiti koliko će biti **vreme njegovog izvršavanja u zavisnosti od broja ulaznih podataka** koje treba obraditi

Notacija veliko O – izražava porast vremena izvršavanja algoritma u najgorem slučaju u zavisnosti od porasta veličine ulaznih podataka

Pored veliko *O* notacije, postoji i notacije malo *o*, omega i teta

Složenost algoritama - primeri

Notacija	Naziv vremena	Primer algoritma
$O(1)$	konstantno	nalaženje većeg od dva broja
$O(\log N)$	logaritamsko	binarno pretraživanje niza
$O(N)$	linearno	linearno pretraživanje niza
$O(N \log N)$	linearno-logaritamsko	najbrži algoritmi za sortiranje, FFT
$O(N^2)$	kvadratno	jednostavni algoritmi za sortiranje
$O(N^3)$	kubno	množenje matrica
$O(C^N), C > 1$	eksponencijalno*	rešavanje TSP putem dinam. programiranja
$O(N!)$	faktorijelno*	rešavanje TSP primenom metode grube sile

*za poslednje dve klase složenosti, ne možemo naći optimalno rešenje za veće instance problema – **kombinatorijalna ekspolozija**! Ovi problemi nisu rešivi u polinomnom vremenu na determinističkoj Tjuringovoj mašini.

Složenost algoritama - primeri

O(*N*) – linearno pretraživanje niza (engl. linear search)

Primer – tražimo broj 2 u sledećem nizu:

6 5 3 1 8 7 2 4

U najgorem slučaju trebaće 8 provera!

Ako znamo da je niz uređen, možemo bolje!

O(log *N*) – binarno pretraživanje niza (engl. binary search)

1 2 3 4 5 6 7 8

U najgorem slučaju trebaće 3 (tj. $\log_2 8$) provere!

Složenost algoritama - primeri

$O(N^2)$ – sortiranje metodom zamenjivanja
(metoda mehura – engl. bubble sort)

Porede se susedni elementi, zavisno od izabranog uređenja (rastuće ili opadajuće), elementi, počev od najvećeg ka najmanjem ili obrnuto, "isplivavaju" na kraj niza, kao mehuri na površinu vode

Algoritam jednostavan za implementaciju, ali uglavnom neefikasan

6 5 3 1 8 7 2 4

Ostali algoritmi za sortiranje: insertion sort, selection sort, quick sort, heap sort, merge sort...

Zanimljiv i koristan link: http://bigocheatsheet.com/

Formalizacija algoritama

Postoji više ekvivalentnih matematičkih formalizacija algoritama (tj. sistema izračunavanja):

- **Tjuringova mašina** (Alan Turing)
- **Lambda račun** (Alonzo Church)
- **Rekurzivne funkcije** (Kurt Gödel)
- **Registarske mašine** (Post-Tjuring mašine – Wang, Minsky, Lambek i još niz naučnika)

Church-Turing-ova teza

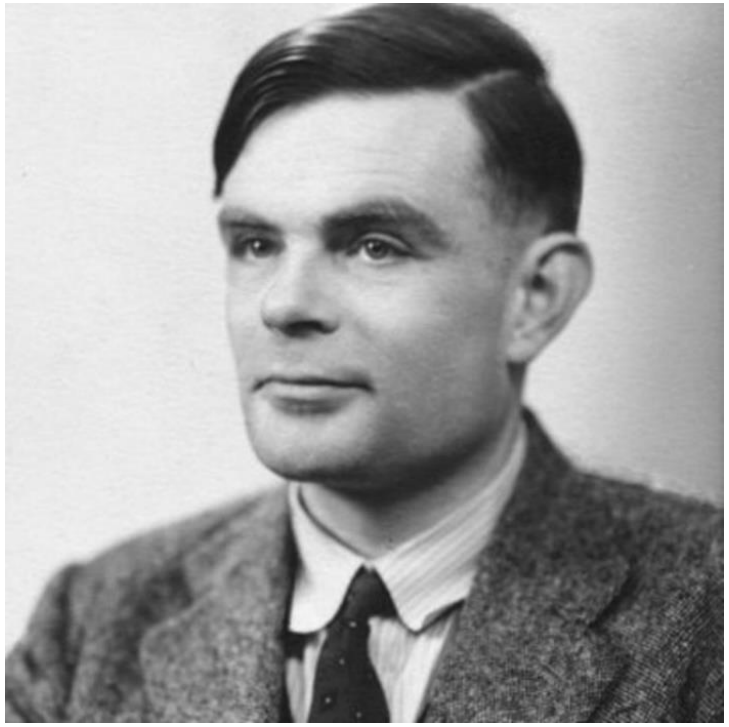
Motivacija: David Hilbert (Entscheidungsproblem 1928.) i Kurt Gödel (teorema o nekompletnosti 1931.), problem zaustavljanja (engl. halting problem)

Problem (funkcija nad prirodnim brojevima**) je algoritamski rešiv** (ignorišući ograničenja resursa**) ako se može rešiti na Tjuringovoj mašini**. Algoritmom se može smatrati svaki niz instrukcija koji se može realizovati na Tjuringovoj mašini.

Tjuringova mašina radi nad konačnim skupom simbola (elemenata) koji se mogu poređati u niz. Dakle, Tjuringova mašina je prebrojiv skup. To znači da je **skup svih algoritama prebrojiv**.

Naravno **skup svih problema odlučivanja je neprebrojiv**, što znači da **postoje problemi za koje se ne mogu definisati algoritmi**

Tjuringova mašina



Alan Turing (1912–1954)

Tjuringova mašina

Tjuringova mašina je **matematički model izračunavanja** koji definiše **apstraktnu mašinu** koja, u skladu sa tabelom operacija, **manipuliše simbolima** koji se nalaze **na traci**. Sastoji se od:

- **beskonačne trake** koja je podeljena na diskretne ćelije; sadržaj svake ćelije može biti specijalni blanko znak ili jedan ili više drugih simbola
- **glave** koja se nalazi nad tačno jednom ćelijom; glava može čitati ili upisivati simbole u svaku ćeliju, kao i pomerati se tačno jednu ćeliju levo ili desno u svakom koraku
- **registra stanja** koji pamti stanje Tjuringove mašine, u svakom trenutku tačno jedno od konačno mnogo mogućih
- **tabele operacija** koja, na osnovu stanja i trenutno pročitanog simbola, kaže mašini da ili obriše ili upiše simbol, pomeri glavu levo ili desno i pređe u novo stanje ili ostane u istom stanju

Približni model Tjuringove mašine

Originalni rad:

Turing, A.M. (1936). "On Computable Numbers, with an Application to the Entscheidungs problem". *Proc. London Mathematical Society*. 2 (pub. 1937). **42**: 230–265.

Neformalni opis Tjuringove mašine

Tjuringova mašina se u svakom trenutku nalazi u jednom od konačno mnogo stanja

- Skup svih stanja se obeležava sa: $Q=\{q_0, q_1, ...\}$
- Mašinom upravlja program koji je sačinjen od konačnog niza naredbi oblika: q_i *s o* q_j
  - *s* je znak nad kojim se nalazi glava
  - *o* je oznaka operacije

Operacije Tjuringove mašine:

- *o* = 1, u ćeliju nad kojom se nalazi glava upiši 1
- *o* = 0, briši sadržaj ćelije nad kojom se nalazi glava
- *o* = L, pomeri glavu jednu ćeliju ulevo
- *o* = R, pomeri glavu jednu ćeliju udesno

Neformalni opis Tjuringove mašine

Primeri naredbi:

q_5 0 1 q_{17} (Ako se mašina nalazi u stanju q_5, a glava nad znakom blanko, u ćeliju se upisuje znak 1 i prelazi u stanje q_{17}.)

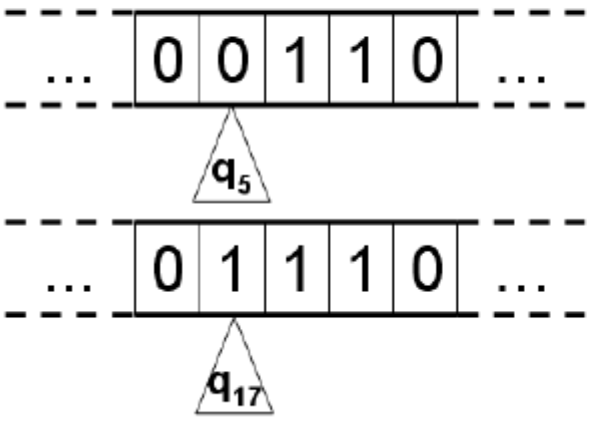
q_1 0 0 q_2 (Ako se mašina nalazi u stanju q_1, a glava nad znakom blanko, u ćeliju se upisuje blanko znak i prelazi u stanje q_2. Ovakva naredba služi samo za promenu stanja mašine.)

q_0 1 L q_0 (Ako se mašina nalazi u stanju q_0, a glava nad znakom 1, glava se pomera ulevo, a mašina ostaje u istom stanju.)

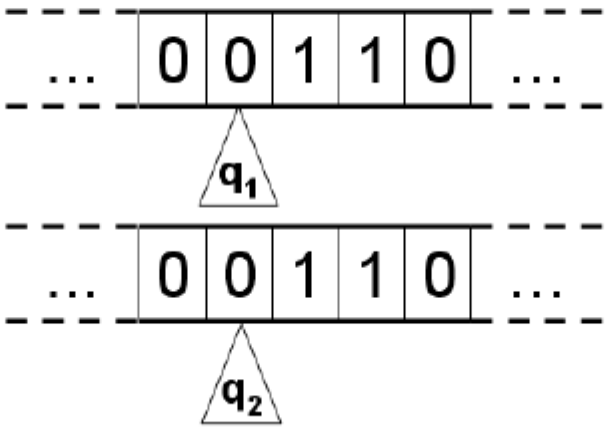
Neformalni opis Tjuringove mašine

Primeri naredbi:

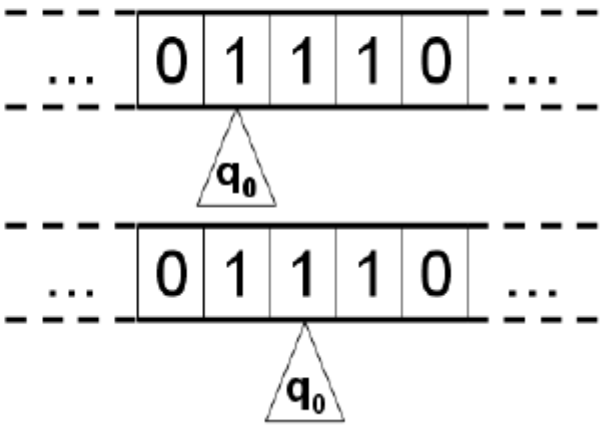
$q_5\ 0\ 1\ q_{17}$



$q_1\ 0\ 0\ q_2$



$q_0\ 1\ R\ q_0$



Neformalni opis Tjuringove mašine

Kada mašina radi **deterministički, program sme sadržati samo jednu naredbu za svaku kombinaciju stanja q_i i sadržaja s ćelije nad kojom je glava.**

Na primer, u jednom programu se ne smeju pojaviti sledeće naredbe:

$q4$ I I $q5$

$q4$ I L $q2$

jer im se vrednosti parametara q_i i s poklapaju, a vrednosti parametara o i q_j razlikuju. Kod **nedeterminističkih mašina** ovaj zahtev **ne postoji!**

Klase složenosti problema:

P – rešivi u polinomnom vremenu na determinističkoj Tjuringovoj mašini

NP – rešivi u polinomnom vremenu na nedeterminističkoj Tjuringovoj mašini, na determinističkoj mašini rešivi samo u eksponencijalnom vremenu

Da li je **P = NP**? Jedno od velikih nerešenih pitanja savremene nauke!

Konvencije kod Tjuringove mašine

Stanje q_0 zovemo **početno stanje**; inicijalno se mašina uvek nalazi u početnom stanju

Traka sadrži konačno mnogo ćelija u koje je upisan znak 1; ostale ćelije su prazne (sadrže znak 0)

Reč se na traci prikazuje kao neprekidan niz ćelija koje sadrže znak 1; sa obe strane reči postoji bar po jedan blanko znak

Na početku i na kraju programa, glava se nalazi iznad prve ćelije koja sadrži znak 1

Završno stanje se obeležava sa q_z

Konfiguracija Tjuringove mašine

Konfiguracija Tjuringove mašine sadrži:

- opis sadržaja trake, položaja glave, stanja mašine

Standardna konfiguracija:

- traka je prazna ili sadrži konačno mnogo nepraznih reči razdvojenih sa po jednim blanko simbolom
- glava je iznad prve ćelije trake koja sadrži znak I
- na početku izvršavanja mašina se nalazi u stanju q_0
- mašina prestaje sa radom kada dođe u stanje q_z

Konfiguracija Tjuringove mašine

Primer:

$q_0\ 1\ R\ q_0$

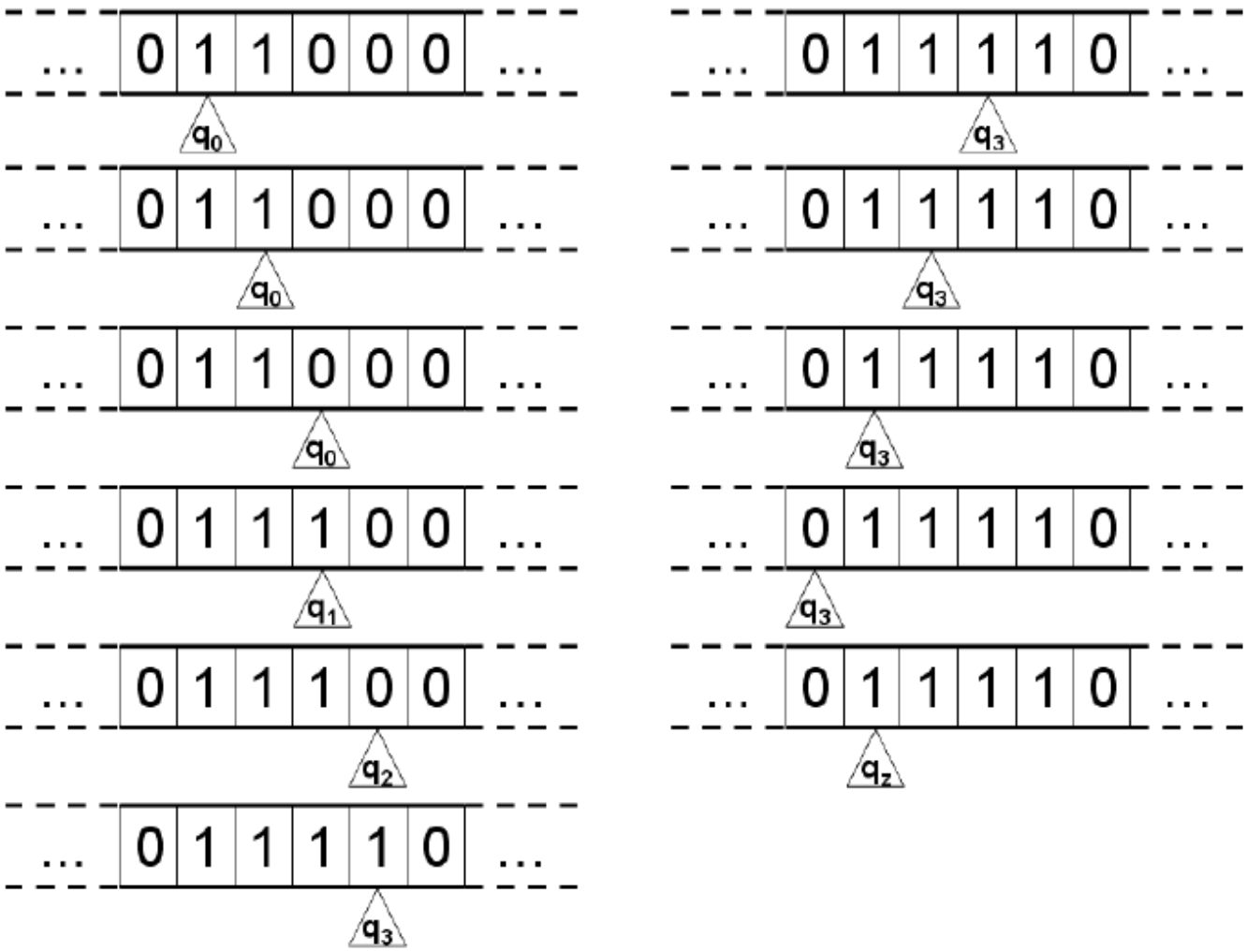
$q_0\ 0\ 1\ q_1$

$q_1\ 1\ R\ q_2$

$q_2\ 0\ 1\ q_3$

$q_3\ 1\ L\ q_3$

$q_3\ 0\ R\ q_z$



Konfiguracija Tjuringove mašine

Šta se dešava ukoliko se glava nađe iznad ćelije za čiji sadržaj ne postoji naredba?

Moguće je program dopuniti naredbama koje ne menjaju ni stanje, ni poziciju glave, ni sadržaj ćelije

Primer: q_0 0 0 q_0

Ova naredba predstavlja **beskonačnu petlju**

Markovljevi algoritmi

Normalni algoritmi Markova ili Markovljevi algoritmi su Tjuring-kompletan sistem za izmenu nizova simbola (stringova) koji koristi pravila nalik gramatičkim. Uvedeni su 1954. od strane Andreja Markova.

Posmatraju algoritam sa stanovišta izvršavanja. Osnovni problem kojim se bave normalni algoritmi Markova je kako prikazati alfabetski operator G na standardni način.

Dve osnovne operacije su **obrada** i **odluka.**

Treća operacija **smena** $\alpha \rightarrow \beta$, kada nam je data neka reč, i podreč α želimo da zamenimo sa β, to radimo po sledećim pravilima:

- Posmatramo reč sleva i kada prvi put primetimo podreč α, na tom mestu vršimo zamenu.
- Primer:
  - pre smene: 49238592349923
  - smena: 923 $\rightarrow$ 0
  - posle smene: 408592349923

Registarske mašine

Četiri najznačajnija tipa u redosledu porasta složenosti modela:

- **Brojačka mašina** – mašina Minskog
- **Pokazivačka mašina** – kombinacija brojačkih i RAM
- **RAM** (engl. Random Access Machine) – brojačka mašina sa indirektnim adresiranjem i proširenim skupom instrukcija
- **RASP** (engl. Random Access Stored Program) – RAM sa instrukcija u registrima, analogna univerzalnoj Tjuringovoj mašini, primer von Neumann-ove arhitekture

Sastoje se od:

- neograničenog broja označenih, diskretnih registara od kojih je svaki neograničenog kapaciteta
- brojača
- (veoma) ograničenog skupa instrukcija
- registra stanja (IR – registar instrukcije)
- liste označenih instrukcija, obično u sekvencijalnom redosledu

Formalni opis sintakse programskih jezika

Sintaksa, semantika i pragmatika

Da bi se program preveo na mašinski jezik nekog računara potrebno je:

- definisati šta su **ispravni programi programskog jezika**
- definisati koja **izračunavanja odgovaraju naredbama programskog jezika**

Pitanjima **ispravnosti** programa bavi se **sintaksa programskih jezika** (i njena podoblast leksika programskih jezika)

Pitanjem **značenja** programa bavi **semantika programskih jezika**

Pitanjima **izražajnosti** bavi se **pragmatika programskih jezika**

Sintaksa i semantika

Sintaksa jezika je skup pravila na osnovu kojih se kreiraju pravilne konstrukcije nekog jezika

- Primenom sintaksnih pravila utvrđujemo da li je određena konstrukcija pravilna
- Sintaksne greške su formalne (pogrešno otkucana reč, spojene dve reči) i prevodilac ih lako otkriva

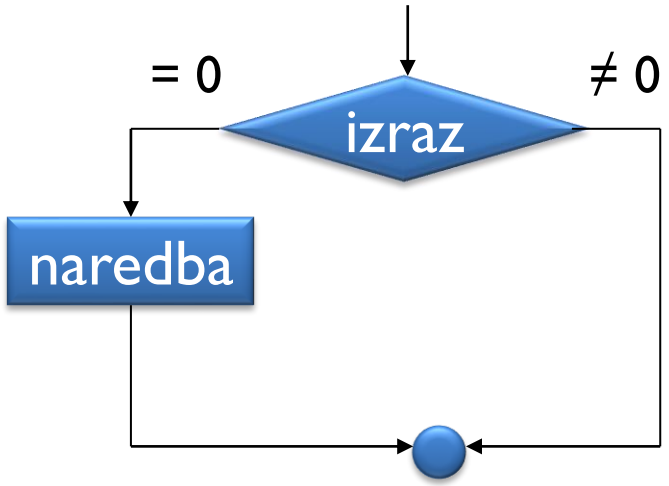
Semantika određuje značenje pojedinih konstrukcija, odnosno programa u celini

- Semantičke greške su vezane za logiku programa i njih otkriva sam programer

Sintaksa, semantika i pragmatika upravljačke strukture nepotpune selekcije

Sintaksa: if (izraz) naredba

Semantika:



Pragmatika: if (x == 0) {y = 1; z = 2;}

Sintaksa programskih jezika

Da bi se definisala ili opisala sintaksa nekog jezika, potrebno je posedovati **specijalan jezik za opis sintakse**

Jezik koji služi za opis nekog drugog jezika naziva se **metajezik**

Metajezici **ne podležu nikakvim međunarodno usvojenim standardima, tako da postoji dosta varijacija u načinu označavanja i korišćenja**

Semantika nekog jezika se najčešće izražava primenom prirodnih jezika

Prirodni jezici služe jedan drugome kao metajezici, jer se svaki jezik može opisivati i definisati sredstvima nekog drugog postojećeg prirodnog jezika

Jezici za opis sintakse

Metajezici za opis sintakse programskih jezika:

- **Bekus-Naurova forma** (BNF)
- **Proširena Bekus-Naurova forma** (EBNF)
- **Sintaksni dijagrami**

Primene: **kompajler-kompajler**, tj. generator kompajlera

- programski alat za kreiranje parsera, interpretera ili kompajlera, na osnovu formalnog opisa jezika i mašine
- ulaz – tekstualni fajl sa gramatikom opisanom u BNF ili EBNF kojom se definiše sintaksa programskog jezika
- izlaz – izvorni kod parsera za programski jezik
- primeri – Atlas, GNU bison

Formalne gramatike

Formalne gramatike definišu tj. generišu formalne jezike – klasifikacija Comskog (tip 0 – rekurzivno prebrojive – Tjuringova mašina, tip 1 – kontekstno osetljive, tip 2 – kontekstno slobodne, tip 3 – regularne – konačni automat). Formalnu gramatiku čine:

- konačan skup završnih (terminalnih) znakova;
- konačan skup nezavršnih (neterminalnih) znakova;
- konačan skup pravila izvođenja čije se leve i desne strane sastoje od nizova takvih znakova
- početni nezavršni znak

Neterminali (nezavršni ili složeni znakovi): metaizrazi koji se moraju dalje objašnjavati (izraz, naredba)

Terminali (završni ili prosti znakovi): metaizrazi koji se ne moraju dalje objašnjavati (if, while, tokeni)

BNF

Bekus-Naurova Forma (BNF) je **notacija za zapisivanje kontekstno slobodnih gramatika** koja služi za opis sintakse jezika koji se koriste u računarstvu (programski jezici, skupovi instrukcija ili komunikacioni protokoli)

BNF metajezik razvijen 1960. prilikom definicije programskog jezika ALGOL 60, poteklog od međunarodnog komiteta kome su bitno doprineli John Backus i Peter Naur

Srodna notacija koristila se u antičko doba za opis sanskrita

BNF notacija - metasimbol <>

Prvi osnovni element BNF notacije je metasimbol zagrada < > koje služe za **definiciju svih složenih objekata** koji se javljaju u posmatranom programskom jeziku (**neterminala**)

Primer: prirodan broj bi označili sa <prirodan broj>

Na sličan način, da bi definisali sintaksu nekog programskog jezika biće nam potrebne precizne definicije objekata kao što su:

- <aritmetički izraz>
- <instrukcija IF>
- <instrukcija WHILE> i mnogi drugi.

Terminali se prikazuju bez ikakvih dodatnih znakova

BNF notacija – metasimbol I

Drugi osnovni element BNF notacije je vertikalna crta I koja označava **ekskluzivnu disjunkciju**

Primer:

<p> | <q>

označava da postoje samo dve mogućnosti: ili će se primeniti objekat <p>, ili će se primeniti objekat <q>

BNF notacija – metasimbol ::=

Treći osnovni element BNF notacije jeste metasimbol **dodele vrednosti** ::=

Simbol dodele vrednosti primenjuje se u metajeziku jer se znak = koristi u programskim jezicima kao relacioni operator, a znak := kao znak dodele vrednosti, pa je bilo neophodno uvesti neki poseban znak za dodelu vrednosti u metajeziku i onda je usvojen ::=

Primer: definiciju

<x> ::= <p> | <q>

interpretiramo tako što kažemo da "objekat x može da bude bilo objekat <p>, bilo objekat <q>" (i ništa drugo izuzev pomenutog)

BNF notacija – lančanje, rekurzija, ponavljanje

Lančanje:

<naredba WHILE>::=(<izraz>)<naredba>

Rekurzija:

<identifikator>::=<slovo>|<identifikator><slovo>|<identifikator><cifra>

Ponavljanje:

BNF notacija se često dopunjava vitičastim zagradama { } koje označavaju ponavljanje obuhvaćenog objekta konačan ili beskonačan broj puta. U opštem slučaju oznakom:

{<x>} *n m*

definišemo da se objekt **<x> ponavlja (tj. lanča) od *m* do *n* puta.**

Ako se izostavi *m* smatra se $m=1$, a ako se izostavi *n* smatra se $n=+\infty$.

EBNF

BNF notacija sa nizom proširenja i modifikacija naziva se **proširena BNF notacija** ili **EBNF** (engl. Extended Backus-Naur Form). Glavna prednost je **bolja čitljivost.**

U većini programskih jezika terminalni simboli navode pod navodnicama ili pod apostrofima

Ako se prethodno usvoji i u metajeziku, onda se nazivi objekata mogu pisati direktno (bez < >)

Umesto ::= može se direktno koristiti običan znak jednakosti (tj. "=" označava jednakost kao deo opisivanog jezika, dok se = koristi u metajeziku).

Znatno se pojednostavljuju neki od iskaza u metajeziku

EBNF notacija

oznaka	značenje
=	jednako po definiciji
|	ekskluzivno ili
.	obavezna oznaka kraja svakog sintaksnog pravila
objekt	neterminalni simbol objekt
“x”	terminalni simbol x
{simbol}	terminalni ili neterminalni simbol se ponavlja 0, I ili više puta
[simbol]	terminalni ili neterminalni simbol se ponavlja 0 ili I put
(A|B)	grupisanje delova u metalingvističkim izrazima

Sintaksni dijagrami

Analitički zapisi sintakse mogu, u slučaju komplikovanih metalingvističkih formula, biti nepregledni

Zapis sintakse u grafičkoj formi pruža preglednije sagledavanje svih alternativa koje postoje u nekoj jezičkoj konstrukciji

Koriste se **sintaksni dijagrami -** usmereni grafovi koji imaju **jednu ulaznu tačku** i **jednu izlaznu tačku** i služe za **definiciju sintaksno ispravnih objekata i konstrukcija programskog jezika**

Svaki sintaksno ispravan objekt definiše se kao jedna od putanja koje kroz sintaksni dijagram idu od ulaza do izlaza

Konstrukcija sintaksnih dijagrama

Svakom pravilu gramatike dodeljuje se **po jedan graf** koji nosi ime **neterminalnog simbola** sa leve strane pravila

Čvorovi grafa su obeleženi simbolima desne strane pravila i to **pravougaonikom** ako odgovaraju **neterminalnom simbolu** gramatike (metalingvističkoj promenljivoj), a **elipsom** ako su pridruženi **terminalnom simbolu** gramatike. Lukovi u grafu povezuju čvorove na način opisan samim gramatičkim pravilom

Jezik predstavljen grafom se dobija **obilaskom puteva u grafu**

Primer: BNF

Jednostavna gramatika za rad sa aritmetičkim izrazima:

<expression> ::= <term> | <expression> "+" <term>

<term> ::= <factor> | <term> "*" <factor>

<factor> ::= <constant> | <variable> | "(" <expression> ")"

<variable> ::= "x" | "y" | "z"

<constant> ::= <digit> | <digit> <constant>

<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Primer: EBNF

Jednostavna gramatika za rad sa aritmetičkim izrazima:

expression = term | expression, "+" , term.

term = factor | term, "*" , factor.

factor = constant | variable | "(" , expression , ")“.

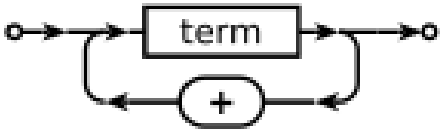
variable = "x" | "y" | "z“.

constant = digit , {digit}.

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9“.

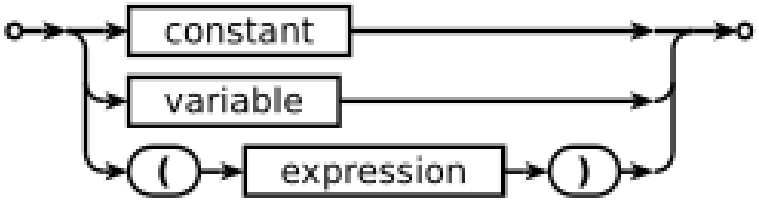
Primer: sintaksni dijagram

expression:

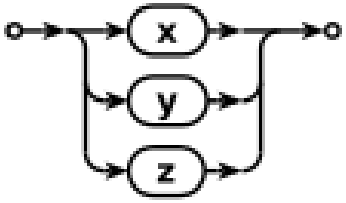


term:

factor:

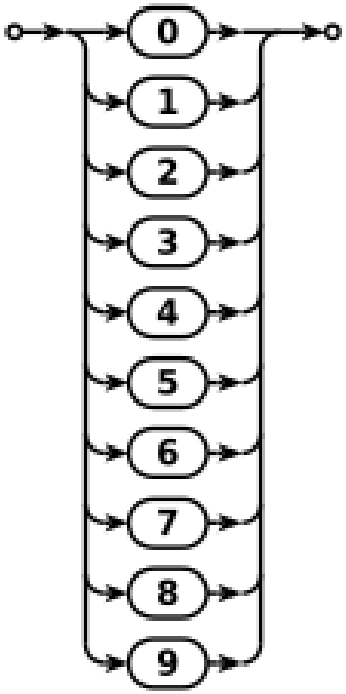


variable:



constant:

digit:



Programski jezik C

Programski jezik C

Dennis Ritchie (1941-2011)

Programski jezik C

Razvijan od 1969. do 1973. u Belovim laboratorijama (Bell Labs)

Jezik 3. generacije, opšte namene, imperativnog stila, u širokoj upotrebi (TIOBE lista za oktobar 2022: 1. Python **2. C** 3. Java)

Inspiracija mnogim programskim jezicima (C++, Java, C#, Go...)

Alat izbora za razvoj operativnih sistema i sistemskog softvera

Osobine:

- Obezbeđuje programeru veliku moć, uz veliku odgovornost!
- Pruža bogat operatorski jezik i kratke sintaksne konstrukcije
- Mali broj ključnih reči, blizak jezicima niskog nivoa, brzi programi
- Fleksibilan i prenosiv, slaba tipizacija, razvijen sistem funkcija
- Bogata standardna biblioteka, integrisan sa UNIX/Linux sistemom

Standardizacija jezika C

K&R C

- opisan u "The C Programming Language" (1978)
- **de facto standard** pre ANSI standarda

C89/C90

- ANSI standard X3.159-1989 (završen 1988; formalno odobren decembra 1989)
- internacionalni standard ISO/IEC 9899:1990

C99

- internacionalni standard ISO/IEC 9899:1999
- uključuje izmene iz Amendment 1 (1995)

C11

- internacionalni standard ISO/IEC 9899:2011

Osobine programskih jezika

Kako bi u programskom jeziku mogli da se opišu svi pojmovi koji se javljaju u algoritmima, neophodno je da on omogući označavanje:

- podataka (konstanti)
- promenljivih
- operacija
- grupisanja operacija
- sekvencijalnog redosleda izvršavanja operacija (sekvence)
- uslovljenog izvršavanja operacija (selekcije)
- ponavljanja izvršavanja operacija (iteracije)

Osnovne sintaksne kategorije

BNF opis jezika C:

https://cs.wmich.edu/~gupta/teaching/cs4850/sumII06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm

cifra: 0 1 2 ... 9

ceo broj: +3 3 -3 0012 156

realni broj:

- **decimalna tačka, a ne decimalni zarez !**

  +7.12 7.12 -7.12 0012.89564

slovo: A B C ... Z a b c ... z

znak: slovo ili cifra ili ! ili ~ ili ... ili ?

karakter: 'a' '\n'

tekst: “I tako dan za danom...“

Azbuka

~ ! # % & * ( ) - _ + = [ ]

{ } : ; ‘ “ ? < > / , \ |

ali i kombinacije

== -- ++ >> <<

plus **rezervisane reči**

Tokeni

Token je **elementarna celina** sačinjena od **simbola azbuke** koja ima neko **značenje** u programskom jeziku

Vrste tokena u jeziku C:

- **rezervisane reči**
- **identifikatori**
- **separatori**
- **konstante**
- **literali**
- **operatori**

Kompajler, nakon faze preprocesiranja, prepoznaje tokene i proverava u sintaksnom analizatoru da li im je redosled navođenja odgovarajući

Rezervisane reči

Rezervisane (službene) **reči** (engl. keyword) su reči koje u programskom jeziku imaju specijalno značenje

U programskom jeziku C postoje samo 32 rezervisane reči

Case sensitive – 'A' i 'a' nisu isti karakter!

auto	**break**	**case**	**char**
const	**continue**	**default**	**do**
double	**else**	**enum**	**extern**
float	**for**	**goto**	**if**
int	**long**	**register**	**return**
short	**signed**	**sizeof**	**static**
struct	**switch**	**typedef**	**union**
unsigned	**void**	**volatile**	**while**

Identifikatori

Identifikator je sekvenca velikih i malih slova, cifara i karaktera "_" (donja crta), koja nije u skupu ključnih reči i obavezno počinje slovom ili donjom crtom. Koriste se kao imena promenljivih, funkcija, makroa, struktura, konstanti.

Primer EBNF:
identifikator = (slovo | "_"){slovo | cifra | "_")

Primeri ispravnih identifikatora:

y upisi_8 _suma Pro_men_lji_va

Primeri neispravnih identifikatora:

suma + 1 512_a $promenljiva while

Identifikatori bi trebalo da su samoopisujući: fun1 ili sumaNiza?

Separatori

Separatori su tokeni predstavljeni specijalnim simbolima koji imaju unapred definisano sintaksno značenje u programu.

Separatori u jeziku C su:

{ }	početak i kraj neke programske celine (bloka)
;	kraj naredbe
( )	početak i kraj liste parametara funkcije
//	linijski komentar
/* */	početak i kraj blokovskog komentara

Konstante

Konstante su konkretne vrednosti napisane direktno u okviru programskog koda.

Vrste konstanti u jeziku C:

- celobrojne
- realne
- znakovne (karakter konstante ili karakteri, pišu se između znakova ‘ ‘).

Konstante za specijalne karaktere u C-u: ‘\n’, ‘\t’, ‘\v’, …

Literali

Literali su konstanti nizovi karaktera

Pišu se između znakova navoda (“) i obično se koriste za prikaz rečenica na govornom jeziku

Literal se mora završiti u jednoj liniji, problem višerednih poruka se rešava pomoću specijalnih karaktera. Primer:

“Ovo je prvi red. \n Ovo je drugi red.”

Operatori

Operator je simbol koji se koristi za označavanje operacije koje je potrebno izvršiti nad operandima

Vrste operatora u jeziku C:

- aritmetički operatori
- relacioni operatori
- logički operatori
- operatori dodele vrednosti
- operatori nad bitovima
- ostali operatori

Od operatora, konstanti i promenljivih formiraju se izrazi

Prioritet operatora određuje redosled izračunavanja

Struktura programa u jeziku C

Primer: "hello, world!" u jeziku C

Prvi primer u K&R:

Uključiti informacije o standardnoj biblioteci

Definisati funkciju pod nazivom main koja ne dobija nikakve vrednosti argumenata

Main poziva bibliotečku funkciju printf za prikazivanje niske znakova
\n predstavlja znak za novi red

```
#include <stdio.h>

main()
{
    printf("hello, world!\n");
}
```

Osnovna struktura C programa

Program se sastoji od jedne ili više funkcija

Funkcije su gradivne jedinice programa i mogu pozivati jedna drugu

Izvršavanje programa počinje od funkcije main()

Osobine strukture C programa:

Linearna u jednom nivou

Top-down razvoj

Prevođenje u jednom prolazu

Prvo definiši, pa tek onda koristi!

Program u jeziku C

Program u jeziku C se sastoji od izvornih datoteka (označenih sufiksom **.c**) i datoteka zaglavlja (označenih sufiksom **.h**)

Prostiji programi obično se pišu u jednoj izvornoj datoteci

Dobra praksa je da se deklaracije funkcija, globalne promenljive, makroi i konstante izdvoje u posebnu datoteku zaglavlja koja se potom pretprocesorskom direktivom **#include** uključuje u izvornu datoteku

Izvorna datoteka sadrži **izvorni** (engl. source) **kod** programa

Između dva susedna tokena može biti proizvoljan broj razmaka i tabulatora

Uobičajeno je novu definiciju ili naredbu započeti u novom redu, takođe uvući ugnježdenu strukturu ili blok naredbi