LEKSIKA, SINTAKSA I SEMANTIKA C++ JEZIKA

Pisanje programa podliježe pravilima koja su slična pravilima pisanja u običnom govornom jeziku.

Programski jezik određuju tri jezičke komponente:

1. **Leksika** – određuje alfabet jezika te kako se tvore leksemi (niz znakova koji čini prepoznatljivu nedjeljivu cjelinu).

2. **Sintaksa** – određuje kojim redom se leksemi slažu u programske iskaze.

3. **Semantika** - određuju značenje programskih iskaza.

Pokazat ćemo kako su definirane u jeziku C++.

Temeljne leksičke kategorije C++ jezika su:

1. **Identifikatori** služe za zapis imena varijabli, funkcija i korisničkih tipova.

2. **Ključne riječi** C++ jezika su: `bool, break, catch, char, class, itd..` Ne smiju se koristiti kao identifikatori jer oni bitno određuju programske iskaze i strukture. Zapisuju se malim slovima.

3. **Literalne konstante** služe za zapis numeričkih i tekstualnih (znakovnih) konstanti (pr. 135, 3.14, 'A', "Hello World").

4. **Operatori** (+,-*/,..=, []..(), &, .+=,*=,...) služe označavanju aritmetičko-logičkih i drugih operacija koje se provode sa memorijskim objektima - varijablama i konstantama.

5. **Leksički separatori** su znakovi koji odvajaju lekseme. Razmak, tabulator i kraja retka tretiraju se kao prazno mjesto, kojim se razdvajaju leksemi. Operatori, također, imaju značaj leksičkih separatora. Znak točka-zarez ('**;**') predstavlja separator - **terminator naredbi.**

6. **Komentar** se piše kao proizvoljni tekst. Dozvoljena su dva načina zapisa :

```
// ovo je komentar
/* i  ovo je
   komentar */
```

7. **Specijalne leksičke direktive** se označavaju znakom **#** na početku retka. One se izvršavaju prije procesa kompajliranja.Primjerice, **`#include<iostream>`** je pretprocesorska direktiva da se u proces kompajliranja uvrsti i sadržaj datoteke imena **`iostream`.**

Elementi programskog jezika

Tipovi i klase	označavaju ‘kategorije’ vrijednosti s definiranim operacijama	`int , float , char`
Konstante	literalni zapis vrijednosti osnovnih tipova	`0 , 123.6 , "Hello"`
Varijable	imena memorijskih lokacija koje sadrže pripadajuće vrijednosti	`i , sum`
Deklaracije	zapisi kojima se nekom imenu pridjeljuje tip (kategorija vrijednosti)	`int x, *px;` `int fun(int);`
Izrazi	zapis proračuna novih vrijednosti kombiniranjem varijabli, konstanti i operatora	`sum + i`
Naredbe	zapisi pridjele vrijednosti i kontrole tijeka programa	`sum = sum + i;` `while, for, if-else, switch, break`
Funkcije	imenovano grupiranje naredbi	`main()` `Hello()`
Moduli (kompilacijska jedinica)	skup međuovisnih varijabli i funkcija, kompajlira se kao jedinstvena cjelina	`datoteka`

Formalni zapis gramatike C-jezika

Navedeni elementi jezika se iskazuju u zapisima (rečenicama) kombinacijom leksema prema strogim gramatičkim (sintaktičkim) pravilima, koji imaju nedvosmisleno značenje.

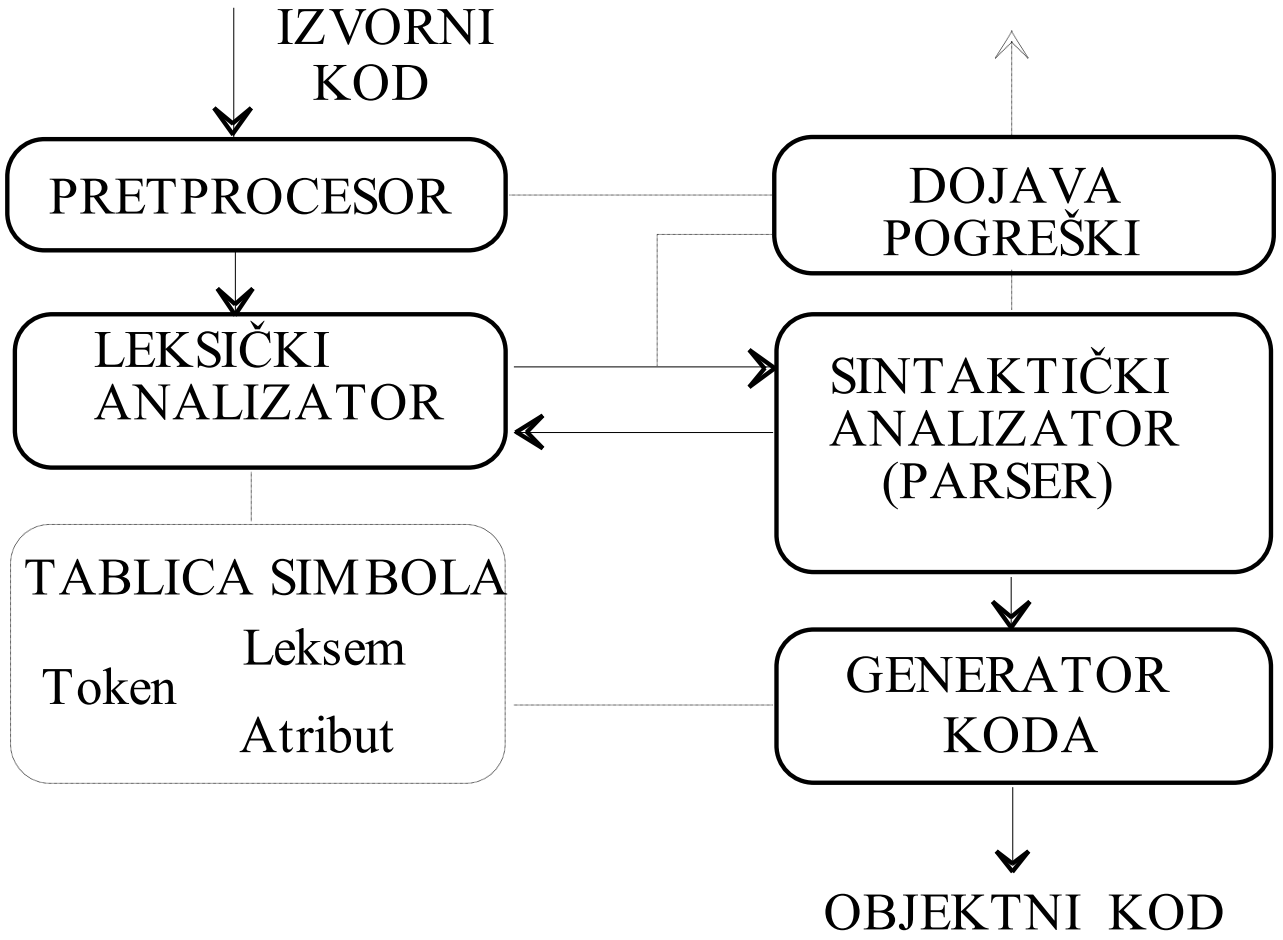
U prirodnim jezicima iskazi mogu imati više značenja (podkontekst), ovisno o razmještaju riječi, o morfologiji (tvorba riječi) i fonetskom naglasku. U programskim jezicima se ne koristi morfološka i fonetska komponenta jezika, pa se gramatika svodi na sintaksu, također, dozvoljen je samo onaj raspored riječi koji daje nedvosmisleno značenje.

Uobičajeno se kaže da gramatika programskih jezika spada u klasu **bezkontesktne gramatike.**

Za opis sintakse nekog jezika koristi se posebni jezik koji se naziva **metajezik**. Jezik koji se opisuje metajezikom naziva se ciljni jezik.

Za opis semantike nekog jezika ne postoje prikladni metajezici već se semantika izražava opisno, primjenom prirodnih jezika.

Proces kompajliranja se odvija na sljedeći način:



1. Izvorni kod može biti spremljenu u jednoj ili u više datoteka koje se u toku jezičkog **pretprocesiranja** formiraju kao jedna datoteka, koja se naziva kompilacijska jedinica.

2. U leksičkoj analizi izvornog koda, se dobavljaju **leksemi**, koji kompajleru predstavljaju **terminalne simbol jezike – tokene:** *ključne riječi*, *specijalni simboli* (operatori i separatori), *identifikatori*, *literalne numeričke i tekstualne konstante*.

3. Sintaktički analizator (parser) dobavlja jezičke simbole i određuje da li su oni grupirani u skladu s definiranom sintaksom. Ukoliko je to zadovoljeno vrši se prevođenje u objektni kod usklađeno sa semantikom jezika.

4. Pogreške u procesu kompajliranja se dojavljuju kao:
    - leksičke pogreške (pr. neispravno zapisano ime varijable)
    - sintaktičke pogreške (pr. u aritmetičkom izrazu nisu zatvorene zagrade)
    - semantičke pogreške (pr. kada je primjenjen operator na dva nekompatibilna operanda)

U programu mogu biti prisutne i logičke pogreške (npr. petlja se ponavlja beskonačno). Njih može otkriti korisnik tek prilikom izvršenja programa.

Primjer Razmotrimo iskaz:

if (a > 3) max = 5.4; **else** max = a;

Ovaj iskaz predstavlja sintaktički ispravan zapis sintaktičkog entiteta - ***IskazaIf***. Odnos leksema, tokena i atributa prikazuje tablica.

Leksem	**kategorija tokena**	**atribut**
"if", "else"	ključna riječ	-
"max", "a"	identifikator	Varijabla
"=", ">"	operatori	-
";"	terminator naredbe	
(...)	separator izraza	
"5.4", "3"	konstanta	numerička vrijednost: 5.4 i 3

Za *Ifiskaz* u C jeziku vrijedi sintaktičko pravilo:

IskazIf "je definiran kao" **if (***Izraz***)** *Iskaz* **else** *Iskaz*
"ili kao" **if (***Izraz***)** *Iskaz*

Ako izneseno sintaktičko pravilo shvatimo kao zapis u nekom sintatičkom metajeziku onda :

IskazIf, *Izraz* i *Iskaz* predstavljaju **metajezičke varijable** koje u odnosu na ciljni jezik predstavljaju **neterminalne simbole**,

Odnosi definiranja: "je definiran kao" i "ili kao" su **metajezički operatori**,

a leksemi: if, then i else i znakovi zagrada su **metajezičke konstante** koje odgovaraju simbolima ciljnog jezika, pa se nazivaju **terminalni simboli ili tokeni**.

Uočimo da "ili kao" operator ima značaj logičkog operatora ekskluzivne disjunkcije.

Sintaktička pravila, kojima se jedan neterminalni simbol definira pomoću niza terminalnih i/ili neterminalnih simbola, nazivaju se **produkcije** jezika.

Prema ANSI/ISO standardu produkcije C-jezika se zapisuju na slijedeći način:

1. Operator "je definiran kao" je zamijenjen znakom dvotočke, a produkcije imaju oblik:
   neterminalni_simbol **:**
   niz terminalnih i/ili *neterminalnih* simbola
2. Alternativna pravila ("ili kao") se pišu u odvojenim redovima.
3. Neterminalni simboli se pišu *kurzivom.*
4. Terminalni simboli se pišu na isti način kao u ciljnom jeziku
5. Opcioni simboli se označavaju indeksom opt (*Simbol*$_{\mathbf{opt}}$ ili Simbol$_{\mathbf{opt}}$).

Zapis produkcije iskaza if-else glasi

```
IskazIf :
     if (Izraz) Iskaz  else Iskaz
     if (Izraz) Iskaz
```

Ovo se pravilo može se napisati i na slijedeći način:

```
IskazIf :
     if (Izraz) Iskaz  ElseIskaz_opt
ElseIskaz :
     else Iskaz
```

U ovom je pravilu uveden je *ElseIskaz* kao opcioni neterminalni simbol. Ako postoji, onda je njegova sintaksa opisana drugim pravilom, a ako ne postoji onda prvo pravilo predstavlja pravilo proste uvjetne naredbe.

Mi ćemo gornja pravila proširiti na način da se operator "ili kao" eksplicitno označava okomitom crtom (|), zbog dva razloga:

1. Na taj način gornja pravila (1-4) su ekvivalentna popularnoj BNF notaciji (BNF notacija je metajezik razvijen 1960. godine prilikom definicije programskog jezika AGOL 60, pri čeme su bitne doprinose dali J.W.Bakus i P.Naur, pa BNF predstavlja kraticu za "Backus-ova normalna forma" ili "Backus-Naur-ova forma").

2. Na taj način se alternativne produkcije mogu pisati u istom redu

Gornjim se pravilima lako može definirati i leksička struktura jezika. Primjerice, temeljni se leksički objekti *znamenka i slovo* mogu definirati pravilima:

```
slovo :A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
      |a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z.

znamenka : 0|1|2|3|4|5|6|7|8|9.

heksa_znamenka : 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|a|b|c|d|e|f.

oktalna_znamenka : 0|1|2|3|4|5|6|7|.
```

Koristeći objekte *znamenka* i *slovo* može se definirati objekt *znak* (koji može biti slovo ili znamenka):

```
znak : znamenka | slovo.
```

Vrlo često potreban element jezika je niz znakova. Njega se definira korištenjem rekurzivne definicije:

```
niz_znakova : znak | niz_znakova znak
```

što se intrepertira ovako: niz znakova je ispravno zapisan ako sadrži samo jedan znak ili ako sadrži niz znakova i s desne strane još jedan znak. Dakle, alternativno pravilo prepoznaje sve nizove koji imaju dva ili više znakova.Treba uočiti da se može napisati i slijedeće:

```
niz_znakova : znak | znak niz_znakova,
```

Identifikatori u C-jeziku (nazivi varijabli, labela, funkcija i tipova) moraju započeti sa slovom ili znakom podvlake '_', pa vrijedi :

```
identifikator :
            slovo
          | _
          | identifikator slovo
          | identifikator znamenka
          | identifikator _
```

Na osnovu ovog pravila, kao ispravno zapisani identifikatori, ocjenjuju se: BETA7 , A1B1 , x , xx , xxx , dok sljedeći zapisi ne predstavljaju indetifikatore: 7, A+B , 700BJ , -beta , x*5 , a=b , x(3).

Pod pojmom liste identifikatora podrazumijeva niz identifikatora međusobno razdvojenih zarezom.

```
lista-identifikatora :
     identifikator
     identifikator , lista-identifikatora.
```

Semantika

Pojedini iskazi, iako gramatički ispravno napisani, trebaju biti dodatno objašnjeni. To se odnosi na:

- **specifičnost primjene naredbe pridjele vrijednosti**
- **pravila prioriteta i asocijativnosti operatora**
- **pretvorbu tipova u aritmetičkim izrazima**

Sasvim općenito, sintaksa naredbe pridjele vrijednosti je:

naredba_pridjele_vrijednosti:
***Lvrijednost* = *Rvrijednost* ;**

Primjer:

```
   a = 5*sin(b)+*px;
*px = *px + 1;
```

Lijeva strane naredbe pridjele vrijednosti često se naziva *Lvrijednost* (*Lvalue*), a desna strana se naziva *Rvrijednost* (*Rvalue*). Dvije su simbolike ovog naziva, prva je da L i R znače lijevo i desno, a druga je da L znači lokacijska vrijednost izraza (adresa u memoriji) a R znači realnu vrijednost izraza (are value).

Na lijevoj strani može biti

- varijabla
- dereferencirani pokazivač

Na desnoj strani su izrazi s aritmetičko-logičkim operacijama, uključujući i poziv funkcije.

Prioritet i asocijativnost operatora

```
   7+7*2*2      * ima veći prioritet od +
= 7+ 14*2       i asocijativan  je s lijeva na desno
= 7+28
= 35
```

Tip operatora	Operator	Asocijativnost
doseg	`::`	**s lijeva na desno**
primarni	`[] . -> ()`	**s lijeva na desno**
postfiks –unarni	`++ --` **typeid cast-op**	**s lijeva na desno**
prefiks – unarni	`++ -- & * + - ~ !` `new delete sizeof`	**s desna na lijevo**
multiplikativni	`* / %`	**s lijeva na desno**
aditivni	`+ -`	**s lijeva na desno**
posmačni	`<< >>`	**s lijeva na desno**
relacijski	`< > <= >=`	**s lijeva na desno**
jednakost	`== !=`	**s lijeva na desno**
bitznačajni "i"	`&`	**s lijeva na desno**
ekskluzivni ili	`^`	**s lijeva na desno**
bitznačajni "ili"	`|`	**s lijeva na desno**
logički "i"	`&&`	**s lijeva na desno**
logički `"ili"`	`||`	**s lijeva na desno**
ternarni uvjetni operator	`?:`	**s desna na lijevo**
pridjela vrijednosti	`= *= /= %= += -=` `<<= >>= &= ^= |=`	**s desna na lijevo**
zarez	`,`	**s lijeva na desno**

Npomena : kompaundni operatori i pridjela vrijednosti djeluju s desna na lijevo:

`b = c = 7` `znači` **`c = 7; b = c;`**

Pretvorba tipova

Automatska pretvorba tipova

Svaki izraz daje neku vrijednost čiji tip ovisi o tipu članova izraza. Kada su u nekom izrazu svi članovi i faktori istog tipa tada je i vrijednost izraza tog tipa.

Za deklaraciju: `float y = 5, x=2;`

izraz **y/x** daje realnu vrijednost `2.5`.

Za deklaraciju: `int y = 5, x=2;`

tada izraz **y/x** daje cjelobrojnu vrijednost 2
(odbacuje se ostatak dijeljenja).

U C jeziku se svi standardni tipovi tretiraju kao numerički tipovi.
Kada u nekom izrazu ima više različitih tipova tada kompajler vrši automatsku pretvorbu tipova. Princip je da se uvijek izvršava jedna operacija s dva operanda. Ako su ta dva operanda različitog tipa onda se prije označene operacije vrši pretvorba tipa niže opsežnosti u tip više opsežnosti. Opsežnost tipa, u redoslijedu od manje prema većoj opsežnosti je:

char → int → unsigned→long → float → double.

Primjerice, za vrijednost varijabli

```
int j=5, k=7;
float x=2.1;
```

izraz :

```
j+7.1*(x+k)
```

se izvršava sljedećim redoslijedom:

1. najprije se vrši proračun izraza u zagradama. U tom izrazu se najprije vrijednost varijable k pretvara (kodira) u format s pomičnim zarezom, jer je drugi operand tipa float. Zatim se toj vrijednosti dodaje vrijednost varijable x.

2. Vrijednost dobijenog izraza se zatim množi s realnom konstantom 7.1, jer množenje ima viši prioritet od zbrajanja.

3. Konačno preostaje da se zbroji vrijednost varijable j s vrijednošću prethodno izračunatog izraza (7.1*(x +k)), koji je realnog tipa. Pošto sada ponovo imamo izraz s dva različita tipa, vrši se pretvorba vrijednosti varijable i u tip float, i tek tada se vri zbrajanje

Zapamti: kada se u izrazima miješaju tipovi int i unsigned, logični rezultat možemo očekivati samo za pozitivne brojeve.

```
unsigned u;
int i = -3;
u = i;
cout << u;
```

bit će ispisano: u= 4294967293.

Zapamti: Pravilo je da se pri pretvorbi realnog u cijeli broj odbacuje decimalni dio. To vrijedi bez obzira koliki je decimalni dio.

```
double d = 7.99;
int i ;
i = d;
cout << i;
```

bit će ispisano 7.

Ekplicitna pretvorba tipova

Ukoliko se ispred nekog izraza ili varijable u zagradama zapiše oznaka tipa, primjerice

```
(float) x
```

time se eksplicitno naređuje C kompajleru da se na tom mjestu izvrši pretvorba vrijednosti varijable x u tip `float`.

> **Zapamti:** Kada se oznaka tipa zapiše u zagradama to predstavlja operator pretvorbe tipa (ili eng. cast operator).

Isti efekt ima tzv. konstruktorski izraz u C++ jeziku:

```
float(x)
```

kojim se specificira konstrukcija objekta tipa `float` iz poznate vrijednosti x.

Primjenu ovog "type cast" operatora ilustrirajmo primjerom.

U sljedećem programu vrši se dijeljenje dva cijela broja i rezultat pridijeluje realnoj varijabli.

```
int main()
{
    int i1 = 100, i2 = 40;
    float f1;
    f1 = i1/i2;
    cout << f1;
    return 0;
}
```

Dobije se ispis: **2.000000**

Pri dijeljenju je izgubljen decimalni dio, iako je rezultat izraza **`i1/i2`** pridjeljen realnoj varijabli. Zašto?

Pravilo je da se pretvorba tipa vrši samo ako se u izrazu nalaze različiti tipovi. Pošto su u izrazu `i1/i2` oba operanda tipa `int` izvršava se dijeljenje s cijelim brojevima. Ako želimo da se sačuva i decimalni dio može se primijeniti operator pretvorbe u jednom od tri oblika:

```
        f = float(i1)/i2;
ili     f = i / float(j);
    ili   f = (float)i / (float)j;
```

Dovoljno je da se pretvorba tipa označi na samo jednom operandu, jer se izrazi računaju tako da se uvijek vrši pretvorba u tip veće opsežnosti.

Proste i strukturalne naredbe C i C++ jezika

Naredbe su iskazi kojima se vrši kontrola tijeka programa. Mogu se klasificirati prema razini apstrakcije računarskog procesa na **proste i strukturalne naredbe**.
Već smo koristili strukturalne naredbe tipa selekcije (if-naredbu), iteracije (while-petlju) i sekvence (niz naredbi).

Sintaksa naredbi C i C++ jezika (prema ANSI standardu)

naredba:
naredbeni izraz
| *složena-naredba*
| *naredba-selekcije*
| *naredba-iteracije*
| *naredba-skoka*
| *označena-naredba*

naredbeni izraz :
$izraz_{opt}$ **;**

složena-naredba :
{ *lista-deklaracija*$_{opt}$ *niz-naredbi*$_{opt}$ **}**

lista-deklaracija :
deklaracija
| *lista-deklaracija deklaracija*

niz-naredbi:
naredba
| *niz-naredbi naredba*

naredba-iteracije:
while (*izraz* **)** *naredba*
| **do** *naredba* **while (** *izraz* **) ;**
| ***for (*** $izraz_{opt}$ ***;*** $izraz_{opt}$ ***;*** $izraz_{opt}$ ***)***
naredba

naredba-selekcije:
if (***izraz* **) *naredba*
| **if (***izraz* **)** *naredba* ***else*** *naredba*
| **switch (** *izraz* **)** *naredba*

naredba-skoka :
goto *identifikator* **;**
| **continue;**
| **break;**
| **return** $izraz_{opt}$ **;**

označena-naredba:
*identifikator****:*** *naredba*
| **case** *konstanti-izraz* ***:*** *naredba*
| **default :** *naredba*