PREVAJANJE PROGRAMSKIH JEZIKOV

Domensko specifičan jezik (DSL)

Datum: 25.05.2025.

Ime skupine: SVD

Člani skupine: Sara Vesković, Vojin Keser, David Jaćović

Cilj naše projektne naloge je razvoj domen-specifičnega jezika (DSL), namenjenega opisu infrastrukture parkirnih mest v Ljubljani. Pri tem bo naš jezik omogočal enostavno in pregledno definiranje stavb, ulic, pripadajočih parkirnih prostorov ter drugih konstrukcij, s čimer bo omogočeno generiranje zemljevida z jasno označenimi lokacijami za parkiranje. Jezik bo omogočal vnos podatkov o ulicah (ime, geografska lega) in parkirnih mestih (lokacija), končni rezultat pa bo mogoče pretvoriti v GeoJSON format za nadaljnjo uporabo. Jezik je zasnovan tako, da je enostaven za uporabo, hkrati pa dovolj zmogljiv, da natančno opiše razporeditev in značilnosti parkirne infrastrukture v mestnem okolju.

Najpre smo izvedli identifikacijo osnovnih entitet v mestu, saj so te entitete temelj za nadaljnje modeliranje in razvoj domensko specifičnega jezika.

Entiteti:

1. City
2. Building
3. Road
4. Park
5. River
6. Tree
7. Junction
8. Parking
9. Marker

Pri čemer:

Blok city predstavlja mesto, ki ga opisujemo. Vsebuje lahko bloke za opis cest, zgradb, parkov, rek, dreves, križišč in označevalcev. To je glavni in osnovni element jezika, saj predstavlja kontekst za vse druge infrastrukturne enote.

```
city  identifier { BLOCKS }
```

Blok building predstavlja zgradbo. Vsebuje lahko ukaze za izris, ki izrišejo obrobo zapolnjenega lika. Obroba mora biti zaprta (začetna in končna točka morata sovpadati), saj se le tako lahko pravilno prikaže zapolnjen objekt.

building identifier {COMMANDS}

Blok road predstavlja cesto. Lahko vsebuje ukaze za izris črt, ki definirajo potek ceste skozi mesto. Uporablja se za prikaz prometnih povezav med različnimi območji.

road identifier {COMMANDS}

Blok park predstavlja parkovno površino ali zelenico. Vsebuje ukaze za izris zapolnjenega območja, pogosto s poudarkom na naravnih ali rekreativnih elementih. Obroba mora biti zaprta, podobno kot pri zgradbah.

park identifier {COMMANDS}

Blok river opisuje potek reke. Ukazi znotraj bloka omogočajo risanje linij ali oblike, ki prikazujejo tok vodne površine. Lahko vključuje tudi širino ali krivulje reke za bolj realistično upodobitev.

river identifier {COMMANDS}

Blok tree predstavlja posamezno drevo ali skupino dreves. Ukazi definirajo pozicijo dreves na karti, pogosto kot točke. Ta element poveča realizem in okoljsko vrednost predstavitve.

tree {COMMANDS}

Blok junction predstavlja križišče cest. Uporablja se za definiranje točk, kjer se več cest sreča ali križa. Lahko vključuje dodatne vizualne oznake za orientacijo na zemljevidu.

junction {COMMANDS}

Blok parking predstavlja parkirno območje. Vsebuje ukaze za izris zapolnjenega prostora, namenjenega parkiranju vozil. Omogoča jasno označevanje parkirišč na zemljevidu.

parking identifier { COMMANDS }

Blok marker se uporablja za postavljanje posebnih oznak na zemljevid, kot so turistične točke, znamenitosti, parkirišča ali druge pomembne lokacije. Markerji so pogosto prikazani kot ikone ali simboli.

marker identifier {COMMANDS}

Potem smo izbrali vse konstrukte, ki jih potrebujemo za definicijo strukture jezika, kot so definicije spremenljivk, blokovne strukture, geometrijski ukazi ter način pisanja koordinat in izrazov.

1. Realna števila - Predstavljajo decimalne številke, uporabljene za koordinate, dimenzije in druge numerične vrednosti (npr. 1 , 3.456, 3.4)

2. Nizi - Besedilne vrednosti, običajno uporabljene kot identifikatorji znotraj jezika (npr. "NIZ")

3. Koordinate (X, Y) - Par števil, ki določa lokacijo točke v prostoru.

4. Bloki (BLOCKS) - Strukturni deli jezika, ki združujejo ukaze za določene entitete, kot so building, road, park…

5. Ukazi - Navodila znotraj blokov:

    5.1. line(POINT, POINT) - Nariše ravno črto med dvema točkama.

    5.2. bend(POINT, POINT, ANGLE) - Nariše ukrivljeno črto z določenim kotom.

    5.3. box(POINT, POINT) - Ustvari pravokotno območje med dvema nasprotnima točkama.

    5.4. circ(POINT, RADIUS) - Ustvari krog s središčem v točki in določenim radijem.

    5.5. polyline(POINTS) - Poveže več točk z zaporednimi črtami.

    5.6 mark(POINT) - Postavi oznako na določeno lokacijo.

6. Konstanta let - Omogoča definiranje konstantnih vrednosti

7. Spremenjljivke - Omogočajo uporabo prilagodljivih vrednosti, ki jih je mogoče računati z izrazi.

8. Matematični izrazi - Podpirajo operacije kot so seštevanje, odštevanje, množenje in deljenje pri definiciji koordinat in dimenzij.

9. Validacija (presečišče med objekti) - Za preverjanje, ali se objekti pravilno sekajo

OPIS TESTNIH PRIMEROV

PRVI PRIMER

Prvi primer prikazuje mesto s štirimi stavbami in štirimi cestami. Stavbe so razporejene v dve skupini, levo in desno, medtem ko ceste vključujejo vertikalne in horizontalne črte ter dva ukrivljena ovinka. Ta primer preverja osnovne elemente, kot so box, line in bend.

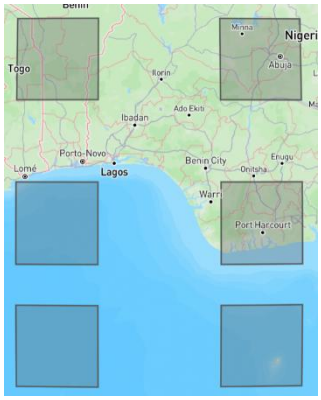
```
city MyCity {
   building A {
      box ( 1.0 , 1.0 , 2.0 , 4.0 )
   } ;
   building B {
      box ( 1.0 , 6.0 , 2.0 , 8.0 )
   } ;
   building C {
      box ( 4.0 , 4.0 , 6.0 , 6.0 )
   } ;
   building D {
      box ( 4.0 , 1.0 , 7.0 , 2.0 )
   } ;
   road a {
      line ( 3.0 , 1.0 , 3.0 , 8.0 )
   } ;
   road b {
      bend ( 3.0 , 4.0 , 1.0 , 5.0 , 90.0 )
   } ;
```

```
  road c {
    line ( 3.0 , 3.0 , 8.0 , 3.0 )
  } ;
  road d {
    bend ( 8.0 , 7.0 , 3.0 , 6.0 , 90.0 )
  } ;
}
```

DRUGI PRIMER

Drugi primer vsebuje šest stavb, razporejenih simetrično v dveh stolpcih, vendar brez cest ali drugih objektov. Namenjen je preverjanju pravilne razporeditve več stavb brez dodatnih motenj.

```
city MyCity {
  building A {
    box ( 1.0 , 1.0 , 3.0 , 3.0 )
  } ;
  building B {
    box ( 1.0 , 4.0 , 3.0 , 6.0 )
  } ;
  building C {
    box ( 1.0 , 8.0 , 3.0 , 10.0 )
  } ;
  building D {
    box ( 6.0 , 1.0 , 8.0 , 3.0 )
  } ;
  building E {
    box ( 6.0 , 4.0 , 8.0 , 6.0 )
```



```
  } ;
  building F {
    box ( 6.0 , 8.0 , 8.0 , 10.0 )
  } ;
}
```

TRETJI PRIMER

Tretji primer je razširjena različica drugega, z dodatkom ene vertikalne ceste, ki povezuje vse stavbe. S tem se preverja, kako se line cesta vključi v že obstoječo razporeditev stavb.

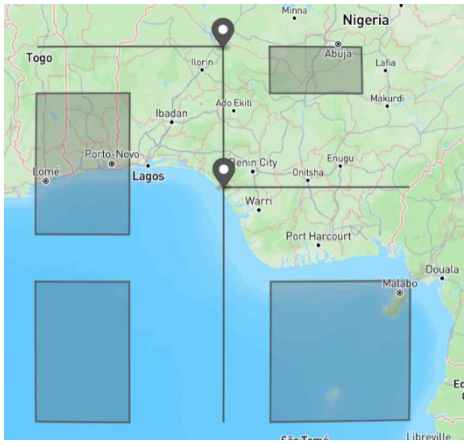
```
city MyCity {
  building A {
    box ( 1.0 , 1.0 , 3.0 , 3.0 )
  } ;
  building B {
    box ( 1.0 , 4.0 , 3.0 , 6.0 )
  } ;
  building C {
    box ( 1.0 , 8.0 , 3.0 , 10.0 )
  } ;
  building D {
    box ( 6.0 , 1.0 , 8.0 , 3.0 )
  } ;
  building E {
    box ( 6.0 , 4.0 , 8.0 , 6.0 )
  } ;
  building F {
```

```
    box ( 6.0 , 8.0 , 8.0 , 10.0 )
  } ;
  road a {
    line ( 4.5 , 1.0 , 4.5 , 10.0 )
  } ;
}
```

ČETRTI PRIMER

Četrti primer prikazuje mesto s uporabo spremenljivke x, štirimi stavbami in tremi cestami. Stavbe so razporejene levo in desno, ceste pa povezujejo zgornje, spodnje in srednje dele mesta. Namenjen je testiranju spremenljivk in kombinacij vodoravnih ter navpičnih cest.

```
let x = 5.0 ;

city MyCity {
  building A {
    box ( 1.0 , x , 3.0 , 8.0 )
  } ;
  building B {
    box ( 1.0 , 1.0 , 3.0 , 4.0 )
  } ;
  building C {
    box ( 6.0 , 1.0 , 9.0 , 4.0 )
  } ;
  building D {
    box ( 6.0 , 8.0 , 8.0 , 9.0 )
  } ;
  road a {
```



```
    line ( 5.0 , 9.0 , 5.0 , 1.0 )
  } ;
  road b {
    line ( 5.0 , 9.0 , 1.0 , 9.0 )
  } ;
  road c {
    line ( 5.0 , 6.0 , 9.0 , 6.0 )
  } ;
}
```

PETI PRIMER

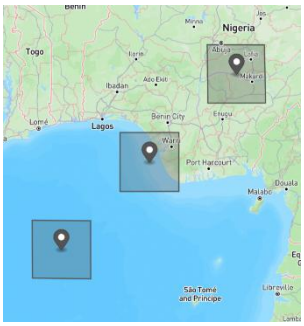
Peti primer prikazuje mesto s parkom in eno cesto, ki ima več točk preloma. Uporabljeni sta dve spremenljivki – ena za številsko vrednost in druga za par koordinat. Namen primera je preverjanje delovanja ukaza polyline ter pravilne uporabe spremenljivk v različnih strukturah.

```
let x = 5.0 ;
let y = new ( 10.0 , 10.0 ) ;
```


```
city MyCity {
  park A {
    box ( 2.0 , 2.0 , y , )
  } ;
  road a {
    polyline ( 2.0 , 2.0 , 5.0 , 3.0 , 7.0 , 8.0 , y , )
  } ; }
```

ŠESTI PRIMER

Šesti primer predstavlja mesto z enim parkom, dvema stavbama in tremi označevalci. Vrednosti koordinat so delno podane s pomočjo spremenljivk in izrazov, kar omogoča preverjanje pravilnega računanja izrazov znotraj ukazov. Namen primera je testirati uporabo aritmetike v definiciji objektov ter vključevanje markerjev za označevanje točk v mestu.

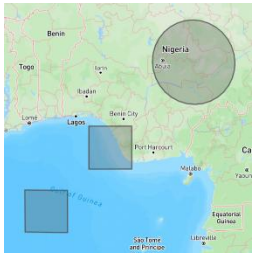



```
let x = 3.0 ;

let y = new ( 10.0 , 10.0 ) ;


city MyCity {
   park A {
      box ( 1.0 , 1.0 , x , x )
   } ;
   building B {
      box ( ( 2.0 + 2.0 ) , ( 1.0 * 4.0 ) , 6.0 , 6.0 )
   } ;
   building C {
      box ( 7.0 , 7.0 , 9.0 , 9.0 )
   } ;
   marker a {
      mark ( 2.0 , 2.0 )
   } ;
   marker b {
      mark ( 5.0 , 5.0 )
   } ;
   marker c {
      mark ( 8.0 , 8.0 )
```

```
    } ;
}
```

SEDMI PRIMER

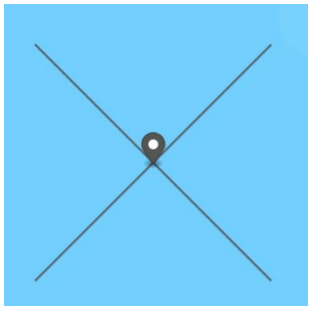
Sedmi primer je podoben šestemu, vendar dodaja krožno stavbo (circ), s čimer se testira nova geometrijska oblika. Cilj je preveriti podporo za krožne objekte poleg pravokotnih.




```
let x = 3.0 ;
let y = new ( 10.0 , 10.0 ) ;

city MyCity {
    park A {
        box ( 1.0 , 1.0 , x , x )
    } ;
    building B {
        box ( ( 2.0 + 2.0 ) , ( 1.0 * 4.0 ) , 6.0 , 6.0 )
    } ;
    building C {
        circ ( 9.0 , 9.0 , 2.0 )
    } ;
}
```

OSMI PRIMER

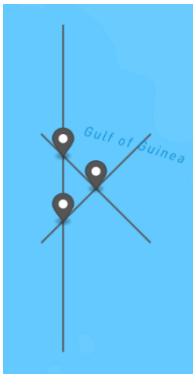
Osmi primer vsebuje izključno eno cesto road A, ki vsebuje dve liniji. S tem se testira možnost več line elementov znotraj ene ceste ter preverja pravilnost (validacija) takšne strukture.



```
let x = 3.0 ;

let y = new ( 10.0 , 10.0 ) ;


city MyCity {
   road A {
      line ( 2.0 , 2.0 , 3.0 , 3.0 )
      line ( 2.0 , 3.0 , 3.0 , 2.0 )
   } ;
}
```

DEVET PRIMER

Deveti primer uporablja spremenljivke in izraze za definiranje več poti. Vključuje seštevanje in druge izraze v koordinatah, s čimer se testira podpora za kompleksnejše izraze znotraj line ter validacija njihove pravilnosti.

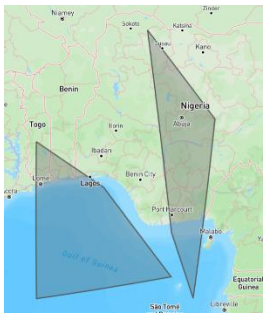



```
let x = 2.0 ;

let y = new ( 3.0 , 3.0 ) ;


city MyCity {
   road A {
      line ( x , x , y , )
      line ( x , ( x + 1 ) , 3.0 , x )
      line ( 2.2 , 1.0 , 2.2 , 4.0 )
```

```
  } ;
}
```

DESETI PRIMER

Deseti primer vsebuje zgradbo z dvema box definicijama, vsaka z več kot štirimi koordinatami. S tem se preverja, ali parser zmore obdelati kompleksne, razširjene definicije boxov in več koordinat v istem objektu.




```
let x = 2.0 ;
let y = new ( 3.0 , 3.0 ) ;


city MyCity {
   building A {
      box ( 1.0 , 1.0 , 7.0 , 2.0 , 4.0 , 6.0 , 1.0 , 8.0 )
      box ( 9.0 , 9.0 , 8.0 , 1.0 , 7.0 , 4.0 , 6.0 , 13.0 )
   } ;
}
```

BNF NOTACIJA

PROGRAM ::= DECLARATIONS CITY_BLOCK

DECLARATIONS ::= LET | epsilon

CITY_BLOCK ::= city CITY

CITY ::= identifier BODY semicolumn

BODY ::= lcurly ELEMENTS rcurly

ELEMENTS ::= ELEMENT ELEMENTS_TAIL

ELEMENTS_TAIL ::= ELEMENT ELEMENTS_TAIL | epsilon

ELEMENT ::= DECLARATIONS BLOCK

BLOCK ::= BLOCK_TYPE_WID identifier lcurly BLOCK_STATEMENT_LIST rcurly semicolumn | tree lcurly BLOCK_STATEMENT_LIST rcurly semicolumn | junction lcurly BLOCK_STATEMENT_LIST rcurly semicolumn

BLOCK_STATEMENT_LIST ::= BLOCK_STATEMENT BLOCK_STATEMENT_LIST | epsilon

BLOCK_STATEMENT ::= DECLARATIONS COMMAND

BLOCK_TYPE_WID ::= building | park | road | river | marker | parking

COMMAND ::= COMMAND_TYPE PARAMETER

PARAMETER ::= lparen ARGUMENTS rparen semicolumn

ARGUMENTS ::= ARGUMENT ARGUMENTS_TAIL

ARGUMENTS_TAIL ::= comma ARGUMENT ARGUMENTS_TAIL | epsilon

COMMAND_TYPE ::= line | bend | box | circ | polyline | mark

ARGUMENT ::= ADDITIVE

ADDITIVE ::= MULTIPLICATIVE ADDITIVE_PRIM

ADDITIVE_PRIM ::= plus MULTIPLICATIVE ADDITIVE_PRIM | minus MULTIPLICATIVE ADDITIVE_PRIM | epsilon

MULTIPLICATIVE ::= UNARY MULTIPLICATIVE_PRIM

MULTIPLICATIVE_PRIM ::= times UNARY MULTIPLICATIVE_PRIM | divide UNARY MULTIPLICATIVE_PRIM | epsilon

UNARY ::= plus PRIMARY | minus PRIMARY | PRIMARY

PRIMARY ::= real_num | variable | lparen ADDITIVE rparen | int

LET ::= SYNTAX VALUE_TYPE semicolumn

VALUE_TYPE ::= new lparen ARGUMENT comma ARGUMENT rparen | ARGUMENT

SYNTAX ::= let variable assign

IZRAČUN FIRST IN FOLLOW MNOŽIC

FIRST (CITY_BLOCK) = {city}

FIRST (CITY) = {identifier}

FIRST (BLOCK_TYPE_WID) = {building, road, park, river,marker,parking}

FIRST (BODY) = {lcurly}

FIRST (PARAMETER) = {lparen}

FIRST (ARGUMENTS_TAIL) = {comma}

FIRST (COMMAND_TYPE) = {line, bend, box, circ, polyline, mark}

FIRST (ADDITIVE_PRIM) = {plus, minus,epsilon}

FIRST (MULTIPLICATIVE_PRIM) = {times,divide, epsilon}

FIRST (UNARY) = {plus, minus, int, variable, lparen,real_num}

FIRST (PRIMARY)= {int, variable, lparen,real_num}

FIRST (SYNTAX) = {let}

FIRST (LET) = {let}

FIRST (MULTIPLICATIVE)= {plus, minus, int, variable, lparen,real_num}

FIRST (ADDITIVE) ={plus, minus, int, variable, lparen,real_num}

FIRST (ARGUMENT) = {plus, minus, int, variable, lparen,real_num}

FIRST (ARGUMENTS) = {plus, minus, int, variable, lparen,real_num}

FIRST (COMMAND) = {line, bend, box, circ, polyline, mark}

FIRST (BLOCK_STATEMENT) = {line, bend, box, circ, polylne, mark,let}

FIRST (BLOCK_STATEMENT_LIST) = {line, bend, box, circ, polylne,mark,epsilon,let}

FIRST (BLOCK) = {building, road, park, river, tree, junction,marker}

FIRST (ELEMENT) = {building, road, park, river, tree, junction, marker,let,parking}

FIRST (ELEMENTS_TAIL) = {building, road, park, river, tree, let,marker, epsilon,junction,parking}

FIRST (ELEMENTS) ={building, road, park, river, tree, junction,marker,let,parking}

FIRST (DECLARATIONS) ={let,epsilon}

FIRST (PROGRAM) = {let, city}

FIRST (VALUE_TYPE) = { plus, minus, int, varriable, lparen, new,real_num}

FOLLOW (PROGRAM) = EMPTY

FOLLOW (CITY_BLOCK) = EMPTY

FOLLOW (CITY) =EMPTY

FOLLOW (DECLARATIONS) = {city,line,bend,box,circ,polyline,mark,building,park,road,river,marker,parking}

FOLLOW (ELEMENTS) = {rcurly}

FOLLOW (BODY) = {semicolumn}

FOLLOW (ELEMENTS_TAIL) = {rcurly}

FOLLOW (ELEMENT) = {building, road, river, park, junction, tree, let, marker, rcurly,parking}

FOLLOW (BLOCK) = {building, road, river, park, junction, tree, let,marker, rcurly,parking}

FOLLOW (BLOCK_STATEMENT_LIST) = {rcurly}

FOLLOW (BLOCK_STATEMENT) = { line, bend, box, circ, polyline, mark, rcurly}

FOLLOW (BLOCK_TYPE_WID) = {identifier}

FOLLOW (COMMAND) = {line, bend, box, circ, polyline, mark}

FOLLOW (PARAMETER) = {line, bend, box, circ, polyline, mark}

FOLLOW (ARGUMENTS) = {rparen}

FOLLOW (ARGUMENTS_TAIL) = {rparen}

FOLLOW (COMMAND_TYPE) = {lparen}

FOLLOW (ARGUMENT) = {comma, rparen, semicolumn}

FOLLOW (ADDITIVE) = {comma, rparen, semicolumn}

FOLLOW (ADDITIVE_PRIM) = {comma, rparen, semicolumn}

FOLLOW (MULTIPLICATIVE) = {comma, rparen, semicolumn, plus, minus }

FOLLOW (MULTIPLICATIVE_PRIM) = {comma, rparen, semicolumn, plus, minus }

FOLLOW (UNARY) = {times, divide, plus, minus, comma, rparen, semicolumn}

FOLLOW (PRIMARY) = {times, divide, plus, minus, comma, rparen, semicolumn}

FOLLOW (LET) = {city,line,bend,box,circ,polyline,mark,building,park,road,river,marker,parking}

FOLLOW (VALUE_TYPE) = {semicolumn}

FOLLOW (SYNTAX) = {plus, minus, int, variable, lparen,new,real_num}