|  |  |
| --- | --- |
| RĪGAS TEHNISKĀ UNIVERSITĀTE | |
| DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE | |
| Informācijas tehnoloģijas institūts | |
|  |  |
| Aleksandrs Koroļko | |
| Programmēšanas valodas C alternatīvas datorgrafikā  Bakalaura darbs | |
|  | |
| Vadības informācijas tehnoloģijas katedra  Rīga, Meža iela 1/3 - 400b, LV-1048, Latvija  Tālrunis 67089594 | Zinātniskais vadītājs:  Artūrs Braučs  Programmētājs |
| Rīga – 2019 | |

RĪGAS TEHNISKĀ UNIVERSITĀTE

DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE

Informācijas tehnoloģijas institūts

APSTIPRINU

Vadības informācijas tehnoloģijas katedras vadītājs

Jānis Grabis

201\_. gada \_\_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bakalaura darba uzdevums

Students: Aleksandrs Koroļko

1. Bakalaura darba tēma:

latviešu valodā: Programmēšanas valodas C alternatīvas datorgrafikā

angļu valodā: Programming Languages C Alternatives in Computer Graphics

Apstiprināta ar ITI direktora 201\_. gada \_\_. \_\_\_\_\_\_\_\_\_ rīkojumu Nr. \_\_\_\_\_\_\_\_\_\_\_

2. Bakalaura darba nodošanas termiņš: 201\_. gada \_\_. \_\_\_\_\_\_\_\_\_

3. Problēmas nostādne:

Noteikt vai ir C programmēšanas valodas alternatīva datorgrafikā, kurai ir priekšroka OpenGL vidē

4. Bakalaura darba mērķis:

Salīdzināt C programmēšanas valodas alternatīvas OpenGL izmantošanas kontekstā

5. Bakalaura darba uzdevumi:

1. Salīdzināt Rust, GO Lang, V Lang, C programmēšanas valodas paradigmas
2. Iepazīties ar Rust, GO Lang, V Lang sintakse un pamatjēdzieniem
3. Iepazīties ar OpenGL vide
4. Izveidot teksturētu objektu ar gaismas atspīdumu renderizācijas programmu OpenGL vidē katrā valodā
5. Salīdzināt daudzobjektu renderizācijas programmas
6. Izveidot 3D pasaules redaktoru uz visefektīvākā valodā

6. Bakalaura darba konsultants (ja nepieciešams):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts, datums)

7. Uzdevuma izsniegšanas datums: 201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bakalaura darba vadītājs: Artūrs Braučs \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts)

8. Uzdevums pieņemts izpildīšanai: 201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(studenta paraksts)

|  |
| --- |
| RĪGAS TEHNISKĀ UNIVERSITĀTE |
| DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE |
| Informācijas tehnoloģijas institūts |
| Programmēšanas valodas C alternatīvas datorgrafikā |
| Aleksandrs Koroļko |
| Anotācija |

Darba mērķi, uzdevumi un risinājumi. Iegūtie rezultāti un to novērtējums.

Darba apjoms - \_\_\_. lpp., \_\_\_ tabulas, \_\_\_ attēli un \_\_\_ pielikumi.

|  |
| --- |
| RIGA TECHNICAL UNIVERSITY |
| FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY |
| Institute of Information Technology |
| Programming Languages C Alternatives in Computer Graphics |
| Aleksandrs Koroļko |
| Annotation |

Goals of the work, results and their estimations.

The work contains\_\_\_ p., \_\_\_ tables, \_\_\_\_ figures and \_\_\_ appendixes.

|  |
| --- |
| [UNIVERSITĀTES NOSAUKUMS OTRAJĀ SVEŠVALODĀ] |
| [FAKULTĀTES NOSAUKUMS OTRAJĀ SVEŠVALODĀ] |
| [Institūta nosaukums otrajā svešvalodā] |
| [Tēmas nosaukums otrajā svešvalodā] |
| [Studenta vārds, uzvārds otrajā svešvalodā] |
| [„Anotācija” otrajā svešvalodā] |

[Darba mērķi, uzdevumi un risinājumi. Iegūtie rezultāti un to novērtējums. Otrajā svešvalodā.]

[Darba apjoms - \_\_\_. lpp., \_\_\_ tabulas, \_\_\_ attēli un \_\_\_ pielikumi. Otrajā svešvalodā.]

**Saturs**

[IEVADS 7](#_Toc32308813)

[1. Teoretiska Daļa 9](#_Toc32308814)

[1.1. Paradigmas salīdzināšanā 9](#_Toc32308815)

[1.2. Rust, GO Lang, V Lang programmēšanas valodas sintakse un pamatjēdzieni 10](#_Toc32308816)

[1.2.1. Programmēšanas valoda Rust 10](#_Toc32308817)

[SECINĀJUMI 23](#_Toc32308818)

[LITERATŪRA 24](#_Toc32308819)

[PIELIKUMI 26](#_Toc32308820)

IEVADS

Mūsdienas C valoda skaitās par vienu no populārākajām programmēšanas valodas, jo lielāka daļa no izveidotas UNIX operētājsistēmas tika bāzēta uz C valodu un tā pat lielāka daļa no jaunajiem projektiem izmanto C valodu kā bāze. Bet jā apskatīt lielāko daļu no mūsdienas programmēšanas valodām, tad visi cenšas apiet to un kļūst labākajai, lai aizvietot C valodu un pārņemt priekšroku izmantojot jaunu sintakse, kas atvieglo kodu saprātošanu un rakstīšanu, kā arī apvienojot dažādas programmēšanas paradigmas, kas palīdz izstrādātājam risināt sarežģītās problēmas ar kuru realizēšanās C valodā būtu vairākas problēmas. No tā ir iemesls apskatīt vai ir kādas no C valodas alternatīviem, kuriem būs priekšrocībā viena no svarīgākajām datorzinātnes nozarē – datorgrafikā, ar kuru ir iespējams rādīt datora veiktspēju.

Ar datorgrafiku ir saistīta lielāka daļa no datorzinātnes nozare kopš no 1951. gadā [1] un mūslaiki šajā nozarē bija sasniegtas lielākie rezultāti, piemērām virtuālas un paplašinātas realitātes realizēšana. Viena no svarīgākajām īpašībām datorgrafika – tai jābūt vairākplatformu programmētai un optimizētai, jo lielāka problēma ir saglabāt kvalitāte un veiktspēju vairākās platformās, no tā bija atrasts mūsdienās risinājums datorgrafikā – OpenGL, kura, salīdzinot ar Windows operētājsistēmas datorgrafikas risinājumu DirectX ir labāk vairākas salīdzinājumos [2], un tai piemīt vairākplatformu izmantošana [3]. Arī OpenGL tas ir atklāta grafiska bibliotēka un no tā tai piemīt realizācija vairākās mūsdienīgas programmēšanas valodas, kuri atšķiras tikai sintaksiski, bet realizācijas pamatideja atstājas, ar ko ir ļoti viegli izmantot standartu OpenGL dokumentāciju [4, 12].

Lai izvēlēt C valodas alternatīvas ir vajadzīgi izvēlēt tādas mūsdienīgas programmēšanas valodas, kuri ir vairāk pietuvināti pie C programmēšanas valoda kompilācijas un pamatdarbības ar datnēm realizēšanas, jo mūsdienas C valoda skaitās par vienu no atraktākajam un sava starpā parastākajam valodām [5], ka arī ir jāizvēlējas tādas alternatīvas, kuriem ir realizēta OpenGL bibliotēka.

Pēc vairāku piemēru apskatīšanu pamatojoties uz valodas informāciju “Slant”[[1]](#footnote-1) tīmekļa resursiem, bija izvēlētas Rust, GO Lang un V Lang programmēšanas valodas, kuri skaitās par tuvākajām C valodas mūsdienīgajām alternatīviem, ka arī, viņiem ir realizētās OpenGL bibliotēkas.

Lai uzzināt, vai no izvēlētajām valodām ir alternatīva, kas ir labāk, nekā C programmēšanas valoda datorgrafikā, pamatojoties uz OpenGL izmantošanu, ir vajadzīgi veikt viena tipa programmas veiktspējas salīdzinājumus uz vairākiem datoriem ar dažādām harakteristikām, lai visas salīdzinājuma posmi bija objektīvi, ka arī ir svarīgi salīdzināt sintakses konstrukcijas, kas ļauj atvieglot vai gan paātrināt koda pamatdarbību, koda lasāmību un kopējo programmas veiktspēju.

# Teoretiska Daļa

Pirms pamat programmas izveidošanai katrā izvēlēta programmēšanas valodā, uz kuras pamata būs veidoti galīgie salīdzinājumi, ir svarīgi zināt Rust, GO Lang, V Lang programmēšanas valodas sintakse un pamatjēdzieni, salīdzināt C Lang, Rust, GO Lang un V Lang paradigmas, ka arī iepazināties ar OpenGL pamat būtne, lai izveidot kvalitatīvu programmu katrā valodā.

## Paradigmas salīdzināšanā

Ar paradigmām ir iespējams uzzināt kā tiek klasificēta programmēšanas valoda un kādām nolūkam to ir jāizmanto. Priekš salīdzināšanai tiek izveidota tabula (1.1 tabula), kurā ir paradītās izvēlētas darba programmēšanas valodas ar visām paradigmām, kuras bija minētās programmēšanas valodas Rust, GO Lang, V Lang dokumentācijā [9-11] un tīmekļa rakstos[[2]](#footnote-2),[[3]](#footnote-3),[[4]](#footnote-4) par kopēju informāciju par paradigmāmH.

1. tabula

**Izvēlētas programmēšanas valodas paradigmas**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C Lang | GO Lang | V Lang | Rust |
| Procedurāla | + | + | + | + |
| Imperatīva | + | + | + | + |
| Objektorientēta | - | + | + | + |
| Vienlaicīga  (Concurrent) | - | + | + | + |
| Strukturētā  (Structured) | + | + | + | + |
| Ģenerisks  (Generic) | - | + | + | + |
| Funkcionāla | - | - | - | + |

Ir redzams, kā lielāka daļa no paradigmas piemīt Rust, GO Lang un V Lang, kas dod priekšrocību virs C Lang programmēšanas valodai, jo ir iespējams izmantojot vienu valodu risināt vairākas problēmas ar vairākiem paradigmas idejām (multi-paradigmas valodas), bet tas varētu kļūst par problēmu, kad tiek izmantotas vairākas paradigmas idejas vienā risinājumā, jo tiek pazemināta koda kvalitāte un tās funkcionalitāte.

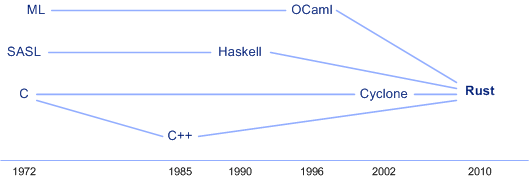
Protams, ka C valodai piemīt mazāka daļa no paradigmas, jo tā ir viena no pirmajiem piemēriem imperatīva paradigmā, uz kura pamatā tika izveidotas vairākas mūsdienīgi zināmās paradigmas pamatojoties uz 1. pielikuma attēlu.

## Rust, GO Lang, V Lang programmēšanas valodas sintakse un pamatjēdzieni

Katra programmēšanas valoda tiek veidota, lai palīdzēt ātrāk un drošāk atrisināt vairākas specifiskas problēmās datorzinātnes un informācijas tehnoloģijas nozarē, kur citās valodās būs sastopamas ar vairākām problēmām, no tā ir svarīgi zināt katrā izvēlēta programmēšanas valoda priekšrocībās pamatojaties uz tās sintakse un pamatjēdzieniem.

### Programmēšanas valoda Rust

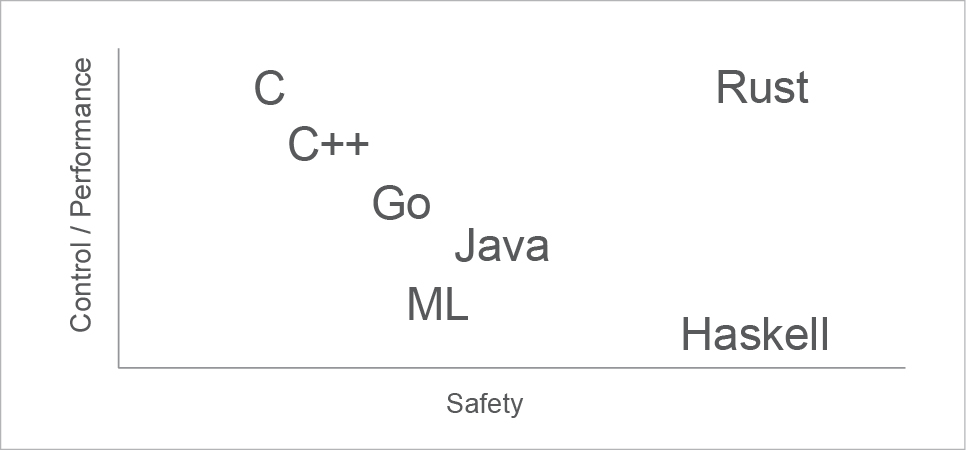
Rust uzskatīts par C Lang tipa programmēšanas valodu, kurā pamatā tiek lietots multi-paradigmas piegājiens, kas savieno sevī vairākas programmēšanas idejas, kā tiek minēts IBM rakstā[[5]](#footnote-5), arī, kā tiek paradīts attēlā[[6]](#footnote-6) (1.1 att.), ir redzams, ka par Rust tuvākajiem priekšgājējiem skaitās OCaml, Haskell, C++ un Cyclone programmēšanas valodas, kas savukārt apvieno tās pamatjēdzienus, sintakse.



1.1 att. **Rust priekšgājēji**

#### Pamatjēdziens

Pamatojoties uz Rust dokumentāciju [9] un uz tīmekļa rakstiem[[7]](#footnote-7),[[8]](#footnote-8), Rust programmēšanas valoda pamatjēdziens apvienot zema līmeņa un augšēja līmeņa programmēšanas valodas jēdzienus, kas rezultātā iekļauj sevī izpildes ātrumu un drošību kā ir paradīts attēlā[[9]](#footnote-9) (1.2 att.) zemāk, bet no tā var arī zaudēties programmas kopējais kompilācijas laiks, kā tiek minēts oficiālā vietnē[[10]](#footnote-10).



1.2 att. **Rust priekšrocības grafiskais salidzinājums**

Rust vidē piemīt mutabilitāte, kas palīdz darboties ar atmiņu drošāk un tīrāk, un viens no pamatjēdzieniem ir sadalīt mainīgus uz nemainīgiem, kas nav tās pats kā konstantes, un mainīgiem, kas tiek definēti ar “mut” pamat vārdu, lai strādāt ar atmiņu izmantojot rādītājus un adreses un nedomāt par neizmantojamām mainīgām, kas varētu aizņemt vietu kaudzē.

Arī Rust vidē ir realizēts pie katra resursa tikai viena saite, kas nodrošina no datnes zaudēšanas, piemēros ir paradītas saistījumi cita mainīgā un funkcijā, kas rezultātā atgriež kļūdu “use of moved value: `v`” “println!” funkcijā, jo vērtība ir pārvietotā citā mainīgā un citā programmas īpašumtiesības kontekstā, kas ir paradīts tabulā lejā (1.2 tabula.).

1. tabula

**Datnes saistījuma kļūdainie piemēri**

|  |  |
| --- | --- |
| Piemērs ar mainīgiem | Piemērs ar funkciju |
| let v = vec![1, 2, 3];  let v2 = v;  println!("v[0] is: {}", v[0]); | fn take(v: Vec<i32>) {  // funkcijas loģika  }  let v = vec![1, 2, 3];  take(v);  println!("v[0] is: {}", v[0]); |

No tā, lai nezaudēt mainīgas vērtību vairākas īpašumtiesības kontekstā ir pieņemts strādāt izmantojot adreses un rādītājus, kur rādītāji apkopo standarta C valodas rādītājus bez papildu marķēšanas vai izpildlaika pārbaudēm, no kas Rust nav vajadzīgs drāzu savācējs, kas paātrina kopdarbību, jo visa analīze tiek veikta sastādīšanas laikā, tādējādi izpildes laikā radot “nulles izmaksas”, kā tiek minēts tīmekļa raksta[[11]](#footnote-11) un dokumentācijā [9] (79. lpp.).

#### Sintakse

Rust sintakse apvieno sevī C Lang un OCaml sintakses, kas rezultāta, ļauj ātrāk iedziļināties C tipa valodas semantikā un sintakse programmētājiem, kuri neizmantoja C tipa valodas, bet ir zināšanās, piemērām, JavaScript valodā. Rezultātā Rust sintaksē ir kombinētās vairākas atslēgvārdu, kas rezultātā ļauj vieglāk kontrolēt vājā tipus mainīgās, gan arī “unsafe” režīmā, struktūras datu kolekcijas, testēšanas pamatfunkcijas, arī vairākas uzlabotas C programmēšanas valodas sintakses konstrukcijas.

<https://doc.rust-lang.org/1.30.0/book/2018-edition/ch02-00-guessing-game-tutorial.html>

Tabulā (1.3 tabula) ir paradītās bāziskas sintakses piemēri ar tās semantiku.

1. tabula

**Sintakses piemēri ar semantiku**

|  |  |
| --- | --- |
| **Sintakse** | **Semantika** |
| let mut x = 5;  x = 10; | Izveidot mainīgo x un piešķirt 5;  Piešķirt 10 pie mainīga x; |
| let x: i32 = 8;  {  println!("{}", x); //Izvada "8"  let x = 12;  println!("{}", x); //Izvada "12"  }  println!("{}", x); // Izvada "8" | Izveidot mainīgo x ar tipu int32 un piešķirt 8;  Atvērt tvērumu;  Izvadīt x; Komentārs “Izvada "8"”  Piešķirt 12 pie x;  Izvadīt x; Komentārs “Izvada "12"”  Ciet tvērumu;  Izvadīt x; Komentārs “Izvada "8"” |
| fn add\_one(x: i32) -> i32 {  x + 1  } | Funkcija “add\_one” ar parametru x int32 tipa un int32 atgriešanās tipu;  Funkcija atgriež x + 1; |
| let y = if x == 5 { 10 } else { 15 }; | Piešķir 10 pie y, jā x ir vienāds ar 5, citādi piešķir 15; |
| struct Point {  x: f64,  y: f64  }  fn main ()  {  let mypoint =  Point {x: 1.0, y: 2.0};  println!(mypoint.x);  } | Deklarēt struktūru ar nosaukumu “Point”, kurā ir lauks x un y ar datu tipu float64;  Funkcijā “main” izveidot jaunu objektu “mypoint” ar “Point” tipu, piešķirt x laukam 1.0, y laukam 2.0;  Izvadīt objekta “mypoint” x lauka vērtību. |
| fn addsub(x: i32, y: i32)  -> (i32, i32)  {  (x + y, x - y)  }  fn main () {  let mut (add, sub)  = addsub(4, 2);  } | Izveidot funkciju “addsub”, ar parametriem x un y, ar int32 tipu, kas atgriež divas int32 vērtībās, kur pirmā vērtība ir x + y un otra vērtība ir x – y;  Funkcijā “main” piešķirt funkcijas “addsub” ar parametriem 4 un 2, pirmo vērtību mainīgajā “add” un otro vērtību mainīgajā “sub”; |

### Programmēšanas valoda GO Lang

GO Lang ir C (C++), Java un Python programmēšanas valodas apvienojums, kurš tiek izstrādāts ar “Google” 2007 gadā, kā tiek minēts GO programmēšanas valoda atskaite[[12]](#footnote-12), kurā pamatideja bija saistīta ar optimālo resursu izmantošanu vairākseržu procesorus un neizmantojot virtuālo mašīnu slāņu, piemēram JVM (Java Virtual Machine), pārveidot cilvēkam lasāmu kodu uzreiz uz bināru kodu procesora vidē, kā tiek minēts tīmekļa raksta[[13]](#footnote-13).

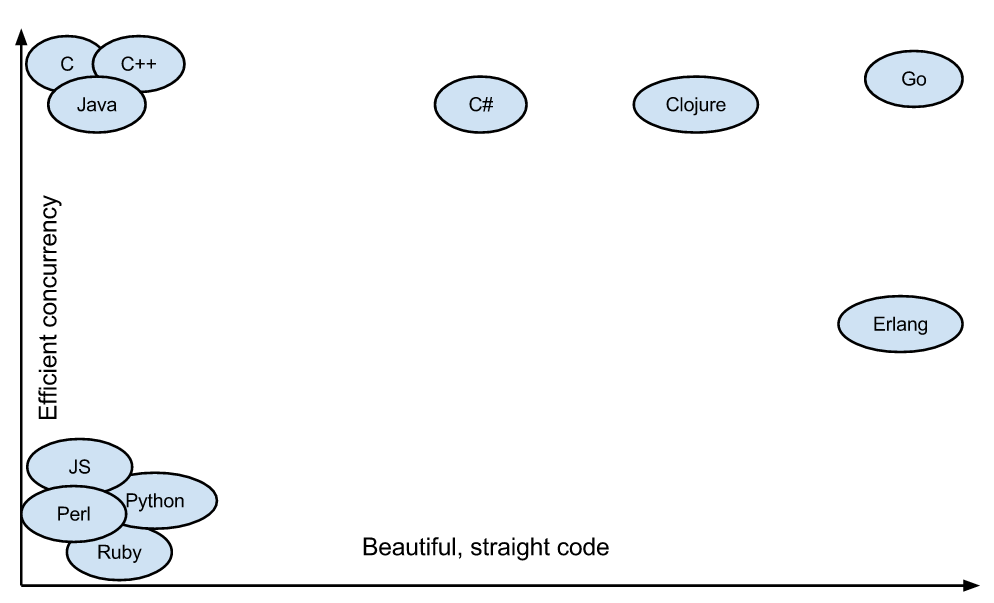
#### Pamatjēdziens

GO Lang valoda pamatjēdziens ir novērst lēnumu un neveiklību daudzkodolu procesora sistēmās, tādējādi padarot dator procesus produktīvāk un mērogojamāk, un atvieglot, nodrošināt, paātrināt pavedienu izveidošanu, samazināt tās glabāšanās vietās daudzumu kaudzē, kas tiek realizēts ar izsekojamām dator resursu vadībām - iebūvētiem pavadoņiem “goroutines”, kā tiek mīnēts un aprakstīts oficiāla apmācības veidnē[[14]](#footnote-14),[[15]](#footnote-15). Arī ir realizēts savs drazu savācējs, kas ļauj neatstāt kaudzē neizmantotas rezervētās datnes.

Kā piemērs, Google izmanto GO Lang servera puses pieprasījumu apstrādāšanai un vairākos projektos, gan arī vecos, kur izmantotas lielas programmatūras sistēmas ar pavedieniem, jo arī ar GO Lang ir ātri un viegli pārtulkot kodu no cita valoda.

Kā tiek parādīts attēlā[[16]](#footnote-16) (1.3 att.), GO Lang skaitās par efektīvāku vienlaicīgumā un vienu no caurspīdīgākām koda sintakse programmēšanas valodām, bet no tā, ka GO Lang sintakse atšķiras no C Lang programmēšanas valodas sintakse ir vajadzīga koda pārtulkošana uz C Lang kompilatora vidē, kas ietekmē uz ilgu kompilācijas laiku.

<https://golang.org/doc/faq#What_compiler_technology_is_used_to_build_the_compilers>



1.3 att. **Vienlaicīgumā un koda caurspīdīgumā valodas grafiks**

#### Sintakse

https://talks.golang.org/2012/splash.article

GO Lang sintakse ir vairāk pietuvināta pie Pascal nekā C Lang, bet tāpat ir izmantotas vairākas konstrukcijas no C Lang. Lielāka atšķirība no citām C Lang ģimenes valodām ir tāda, ka GO Lang sintakses vidē ir tikai 25 atslēgvārdu, bet piemērām C99 ir 37 un C++11 ir 84 atslēgvārdu skaits un šīs skaits palielinās, un daļēji GO Lang semantika ir atvieglota salīdzinot ar C valodas tipa semantiku, piemērām : nav rādītāja aritmētikas, nav netieša skaitliska konvertācija, ir likumīgi ņemt kaudzīta mainīgā adresi utt., kas ļauj saprasts GO Lang sintakse ātrāk un no tā, ļauj jauniem programmētājiem ātrāk iedziļināties C veida programmēšanas valodas vidē.

Tabulā (1.4 tabula) ir paradītās bāziskas sintakses piemēri ar tās semantiku.

1. tabula

**Sintakses piemēri ar semantiku**

|  |  |
| --- | --- |
| **Sintakse** | **Semantika** |
| var x string = "Hello World"  fmt.Println(x) | Izveidot mainīgo x String tipa un piešķirt “Hello World”;  Izvadīt x mainīga saturu konsolē; |
| var arr [5]int  x[4] = 100  fmt.Println(x) | Izveidot piecu int tipa elementu masīvu arr;  Pievienot pie arr piektā elementa 100;  Izvadīt masīvu arr konsolē; |
| func add\_one(x int) int {  return x + 1  } | Funkcija “add\_one” ar parametru x int tipa un int atgriešanās tipu;  Funkcija atgriež x + 1; |
| func main() {  addsub := func(x, y int) int {  return x + y, x - y  }  fmt.Println(addsub(5,6))  } | Izveidot funkciju ar parametriem x un y, int tipa, kura atgriež divas int vērtībās, kur pirmā vērtība ir x + y un otra vērtība ir x – y, main funkcijā;  Ievadīt jaunajā funkcijā 5 un 6 un izvadīt konsolē; |
| type Point struct {  x, y float64  }  func main() {  var mypoint Point  mypoint := Point{1.0, 2.0}  fmt.Println(mypoint.x)  } | Deklarēt struktūru ar nosaukumu “Point”, kurā ir lauks x un y ar datu tipu float64;  Funkcijā “main” izveidot jaunu objektu “mypoint” ar “Point” tipu, piešķirt x laukam 1.0, y laukam 2.0;  Izvadīt objekta “mypoint” x lauka vērtību. |
| func example(n int) {  for i := 0; i < n; i++ {  fmt.Println(i)  }  }  func main() {  go example(5)  } | Izveidot metode example ar paramentru n int tipa, kur ar ciklu izvada visus ciparus no 0 līdz n;  Funkcijā main izmantojot pavedienu izsaukt metode example ar n vērtību 5; |

<https://kuree.gitbooks.io/the-go-programming-language-report/content/2/text.html>

https://golang.org/doc/faq#Is\_Go\_an\_object-oriented\_language

### Programmēšanas valoda V Lang

V Lang – tas ir uz GO Lang uzrakstīta un iedvesmotā ar Rust, Swift, Oberon, jauna programmēšanas valoda, kura apvieno sevī vairākas C tipa programmēšanas valodas pamatjēdzienus un priekšrocības, un pievieno jaunas C tipa valodas iespējās, kā tiek paradīts tabulā (1.5 tabula) lejā.

1.5 tabula

**Valodas iespējas un priekšrocības**

|  |  |
| --- | --- |
| **Priekšrocības un iespējas** | **Valodas** |
| Ātra kompilācija | D, Go, Delphi |
| Vienkāršība un uzturamība | Go |
| Lieliska veiktspēja, salīdzinot ar C un nulles izmaksas C interop | C, C++, D, Delphi, Rust |
| Drošība | Rust |
| Viegls vienlaicīgums | Go |
| Šķērskompilācija | Go |
| Laika koda ģenerēšana un kompilēšana | D |
| Mazais kompilators ar nulles atkarībām | - |
| Nav globāla stāvokli | - |
| “Hot code” pārlādēšana | - |

Pēdējās trīs rindas nav minētas valodas, ar to tiek paradīts, ka pagaidām nav C tipa valodas ar sekojošam iespējām un priekšrocībām, bet V Lang iekļauj sevī arī tos.

Pagaidām V Lang ir Beta versijas stāvoklī un pilna v1.0 versija tiek ieplānotā 2020. gadā beigās, bet jau V Lang tiek izmantots un testēts vairākas servera daļa, WEB, datorgrafikas, kross platformas, GUI, programmatūrās, ka arī V Lang vidē tiek piedāvāts C Lang pārtulkotājs, kurš var pārtulkot kodu no V Lang uz C Lang un otrādāk.

#### Pamatjēdziens

Galvenais pamatjēdziens V Lang ir kompilatora paātrināšana un mazāka vietu aizņemšana, gan arī bibliotēkām, kā tiek paradīts tabulā (1.6 tabula) lejā, un ir redzams, ka V Lang kompilators ar pamat bibliotēkām aizņem vismazāk atmiņu un tiek uzcelts sistēmā ātrāk nekā citi, bet V Lang kompilators ir bāzēts uz C Lang kompilatora un izmanto vairākas C Lang bibliotēkas, un no tā ir svarīgi instalēt arī C Lang.

1. tabula

**Kompilatora un pamat bibliotēkas aizņemta vieta un uzcelšanas laiks salīdzināšana**

|  |  |  |
| --- | --- | --- |
| Kompilators | Aizņemta vieta | Uzcelšanas laiks |
| GO Lang | 525 MB | 1m 33s |
| Rust | 30 GB | 45m |
| GCC | 8 GB | 50m |
| C Lang | 90 GB[[17]](#footnote-17) | 25m |
| Swift | 70 GB[[18]](#footnote-18) | 90m |
| V Lang | < 2 MB | 0.09s – 1s |

Otrs V Lang pamatjēdziens ir uzlabot un apvienot vienā vidē vairākas C tipa valodas iespējas un pamatjēdzienus un kļūst ātrāk, vieglāk, drošāk nekā C Lang un jau tagad V Lang ļauj kompilēt ātrāk nekā citas C tipa valodas, jo kā tiek minēts dokumentācijā, V Lang var kompilēt no 100 tūkstošiem līdz 1.2 miljoniem koda rindas sekundē (uz datora ar sekojošo dzelzi : Intel i5-7500 @ 3.40GHz, SM0256L SSD), jo programma tiek kompilēta bez jebkādām atkarībām uzreiz bināra kodā.

V Lang tiek vairāk atbalsta GUI un datorgrafikas programmatūrās un priekš sekojošiem jau ir izveidotas sākum bibliotēkas, jo ar “Hot code” pārlādēšanu visas koda izmaiņas tiek kompilētas uzreiz, kas ir svarīgi, kad vajadzīgs izmainīt mazu koda daļu un neatjaunot visu programmu.

#### Sintakse

V Lang sintakse ir apvienojums no GO Lang, Rust, Swift, Oberon un C++ sintakses, kur vairākas sintakses konstrukcijas ir uzlabotas, paātrinātas un atvieglotas. Kopumā V Lang sintakse apvieno sevi GO Lang vienkāršību un vairākas C tipa valodas labākus un elastīgus risinājumus.

Tabulā (1.7 tabula) ir paradītās bāziskas sintakses piemēri ar tās semantiku.

1. tabula

**Sintakses piemēri ar semantiku**

|  |  |
| --- | --- |
| **Sintakse** | **Semantika** |
| mut age := 5  age = 10 | Izveidot mainīgo x un piešķirt 5;  Piešķirt 10 pie mainīga x; |
| mut arr := ['Ieva']  arr << 'Peteris'  println(arr.len)  println('Henrijs' in arr) | Izveidot masīvu “arr” ar “Ieva” vērtību;  Pievienot masīvā “Peteris”;  Izvadīt konsolē masīva garumu;  Izvadīt konsolē pārbaudījumu vai ir “arr” masīvā elements “Henrijs”; |
| y := if x == 5 { 10 } else { 15 } | Piešķir 10 pie y, jā x ir vienāds ar 5, citādi piešķir 15; |
| struct Point {  x f64  y f64  }  fn main () {  p := Point{  x: 1.0  y: 2.0  }  println(p.x)  } | Deklarēt struktūru ar nosaukumu “Point”, kurā ir lauks x un y ar datu tipu float64;  Funkcijā “main” izveidot jaunu objektu “mypoint” ar “Point” tipu, piešķirt x laukam 1.0, y laukam 2.0;  Izvadīt objekta “mypoint” x lauka vērtību. |
| fn addsub(x, y int) (int, int) {  return x + y, x - y  }  fn main () {  add, sub := addsub(4, 2)  } | Izveidot funkciju “addsub”, ar parametriem x un y, ar int32 tipu, kas atgriež divas int32 vērtībās, kur pirmā vērtība ir x + y un otra vērtība ir x – y;  Funkcijā “main” piešķirt funkcijas “addsub” ar parametriem 4 un 2, pirmo vērtību mainīgajā “add” un otro vērtību mainīgajā “sub”; |
| fn sqr(n int) int {  return n \* n  }  fn numbers(n int) int {  mut sum := 0  mut i := 0  for i <= n {  sum += i  i++  }  return sum  }  fn run(value int, op fn(int) int) int {  return op(value)  }  fn main() {  println(run(5, sqr))  println(run(5, numbers))  } | Izveidot funkciju “sqr”, ar parametru “n” int tipa, kas atgriež int tipa rezultātu n \* n;  Izveidot funkciju “numbers\_sum”, ar parametru “n” int tipa, kas atgriež int tipa summu no 0 līdz n;  Izveidot funkciju “run”, ar parametriem “value” (int tipa), “op” funkcijas nosaukums ar vienu int parametru, kas atgriež int vērtību, kurā tiek izsauktas funkcijas un atgrieztas tās vērtības;  Funkcijā “main” izvadīt rezultātus “sqr” un “numbers\_sum”, izmantojot funkciju “run”, kur “value” ir vienāda ar 5; |

https://vlang.io/docs#for

<http://vlang.com/docs/>

<https://vlang.io/docs#introduction>

## OpenGL pamati

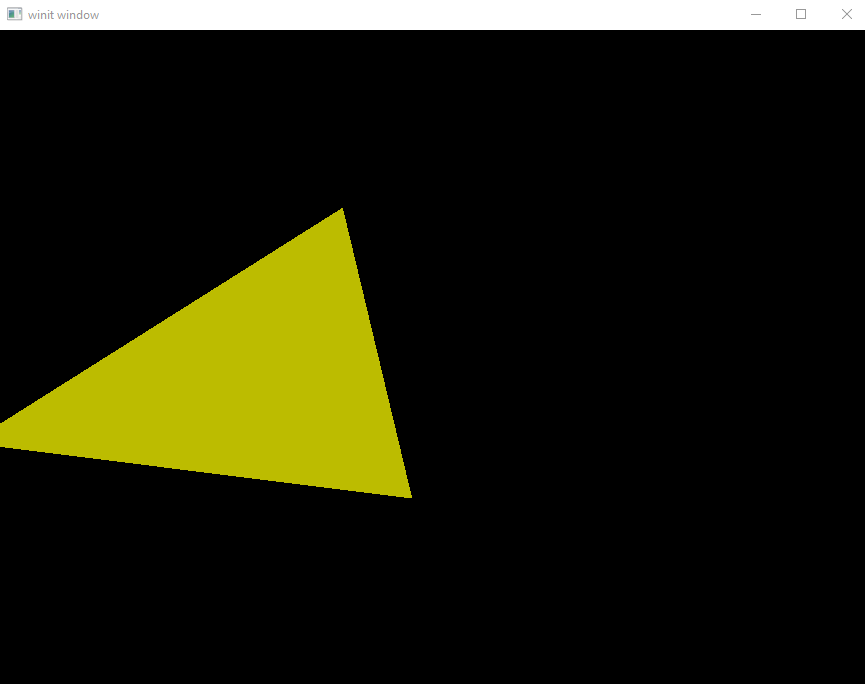
OpenGL (“Open Graphics Library”) ir API (lietojumprogrammu interfeiss) grafikas aparatūrai, kurš sastāv no vairāku simtu funkciju un procedūru kopuma, kas ļauj norādīt ēnotāju programmas, objektus un darbības, kas saistītas ar augstas kvalitātes grafisko attēlu veidošanu. Sākot no OpenGL 4.3 versijas gandrīz visas operācijas, matemātiskas darbībās, datu un attēla kadra buferi glabāšana tiek realizēta ar GPU (grafiskais procesors) atmiņu un ēnotāju programmas izpilde tiek pārnesta uz GPU procesiem, kas kopumā paātrina grafiskas zīmēšanas procesus, bet arī tiek atstāta CPU izmantošana, kad GPU veiktspēja ir maza.

Šajā nodaļā ir apskatītas OpenGL izpildes modelis un ēnotāju programmas jēdziens.

### Izpildes modelis

OpenGL izpildes modelis nodrošina tiešu 3D un 2D grafikas pamatfunkciju kontroli, kā piemērām, primitīvas manipulēšana, parametru precizēšanu ēnotāju programmās, kropļojumnovērse utt., bet OpenGL nenodrošina līdzekļus sarežģītu ģeometrisko objektu aprakstam vai modelēšanai, no kā ir vajadzīgi renderēt lielu sarežģītu objektu no vairākiem primitīvas grupām.

OpenGL vidē pamat zīmēšanas objekti ir primitīvas (punkts, līnijas segments, daudzstūris, lauciņš) un tās zīmēšana tiek realizēta ar ēnotāju programmām un fiksētu funkciju apstrādes vienībām. Visus primitīvus definē viena vai vairāku virsotņu grupa, kur viena virsotne ir punkts vai galapunkts līnijā, vai daudzstūra stūri, kur satiekas divas malas. Katra virsotne tiek apstrādāta neatkarīgi no citām virsotnēm un katrai virsotnei ir iespējams definēt tās pozīcijas koordinātas, krāsas, normālu (perpendikulārs vektors), tekstūras koordinātas utt. OpenGL logā (1.4 att.).



1.4 **OpenGL logs**

OpenGL komandu izpildes interpretācijas modelis ir bāzēts uz klients-serveris modeli, kur klients ir programma, kura izdod komandas, un serveris ir OpenGL, kurš apstrādā un interpretē komandas un pārveido uz dzelža līmenī. Visas komandas vienmēr tiek apstrādātas saņemšanas secībā, lai arī pirms komandas seku realizācijas var būt nenoteikts kavējums, piemērām, viens primitīvs ir jānozīmē pilnībā, pirms jebkurš nākamais var ietekmēt kadru buferi, vai arī vaicājumu un pikseļu lasīšanas operācijas atgriežas stāvoklī, kas atbilst visu iepriekš izsaukto OpenGL komandu pilnīgai izpildei, izņemot gadījumus, kad skaidri norādīts otrādāk.

### Ēnotāju programmas jēdziens

Mūsdienu OpenGL renderēšanas process ir ļoti atkarīgs no ēnotāju izmantošanas, ka piemērs ir iespējams izpildīt tikai loga notīrīšanu neizmantojot ēnotājus, no tā ēnotāji ir viena no svarīgākiem elementiem OpenGL vidē ar kuriem ir iespējams veikt primitīvas matemātiskas pārveidošanas, apgaismošanas, teksturēšanas, ēnošanas operācijas utt.

OpenGL objekta grafiska apstrāde tiek sadalīta četros apstrādes posmos un viena aprēķināšanas posma, kurus kontrolē sava nodefinēta ēnotāja programma :

1. Virsotnes ēnojuma (The Vertex shading) posms – saņem virsotnes datus no virsotnes bufera objektos, apstrādājot katru virsotni atsevišķi. Šis posms ir obligāts visām OpenGL programmām, un tai jābūt saistītai ar ēnotāju.
2. Flīzēšanas ēnojuma (The Tessellation shading) posms – nav obligāts posms, kurā ir iespējams ģenerēt papildu ģeometriju priekš objekta virsotnēm.
3. Ģeometrijas ēnojuma (The Geometry shading) posms – nav obligāts posms, kurā ir iespējams modificēt veselus ģeometriskos primitīvus, ka piemērām mainīt ģeometriskās primitīvas veidu (piemēram, daudzstūrus pārveidojot par līnijām).
4. Fragmenta ēnojuma (The Fragment shading) posms - šajā posmā tiek apstrādāti atsevišķi fragmenti, kurus noģenerē OpenGL rasterizators un tiek aprēķinātas fragmenta krāsas un dziļuma vērtības, un pēc tam nosūtīts tālāk fragmenta pārbaudes apstrāde.
5. Aprēķināšanas ēnojuma (The Compute shading) posms – nav obligāts posms, kurā realizēts kadru bufera pēcapstrāde vai citas darbības ar renderētiem primitīviem, piemēram, statiskus objektus saglabāšana kešatmiņā.

Ēnotāju programmām ir sava sintakse un OpenGL izmanto savu GLSL valodu, kas ir līdzīgs C Lang sintaksei, tabulā (1.8 tabula) lejā tiek paradītas ēnotāju programmas piemēri.

1. tabula

**Ēnotāju programmas kods un apraksts**

|  |  |
| --- | --- |
| Kods | Apraksts |
| #version 140  out vec4 color;  void main() {  color = vec4(0.5, 0.5, 0.0, 1.0);  } | Fragmenta ēnojuma programma, kurā tiek realizēta objekta pārzīmēšana dzeltenā krāsā. |
| #version 140  in vec2 position;  uniform mat4 matrix;  void main() {  gl\_Position = matrix \* vec4(position, 0.0, 1.0);  } | Virsotnes ēnojuma programma, kurā tiek realizēta objekta pārvietošana izmantojot transformācijas matricas reizinājums. |

SECINĀJUMI

LITERATŪRA

1. Chris Woodford, *Computer Graphics,* 10. septembrī 2018. gads [skatīts 14. decembrī 2019. gadā] Pieejams: <https://www.explainthatstuff.com/computer-graphics.html>

2. Autora nav, *OpenGL vs. DirectX: A Comparison,* [skatīts 14.decembrī 2019. gadā] Pieejams: https://www.cprogramming.com/tutorial/openglvsdirectx.html

3. Dmitry Guzeev, *What is OpenGL and how to start learning it?,* 19. janvāris 2018. gads [skatīts 14. decembrī 2019. gadā] Pieejams: <https://medium.com/@dmitrygz/what-is-opengl-and-how-to-start-learning-it-34f19cfa219f>

4. Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane, *OpenGL Programming Guide Eighth Edition*, Michigan, 2013. gads, 986 lpp Pieejams: https://www.cs.utexas.edu/users/fussell/courses/cs354/handouts/Addison.Wesley.OpenGL.Programming.Guide.8th.Edition.Mar.2013.ISBN.0321773039.pdf

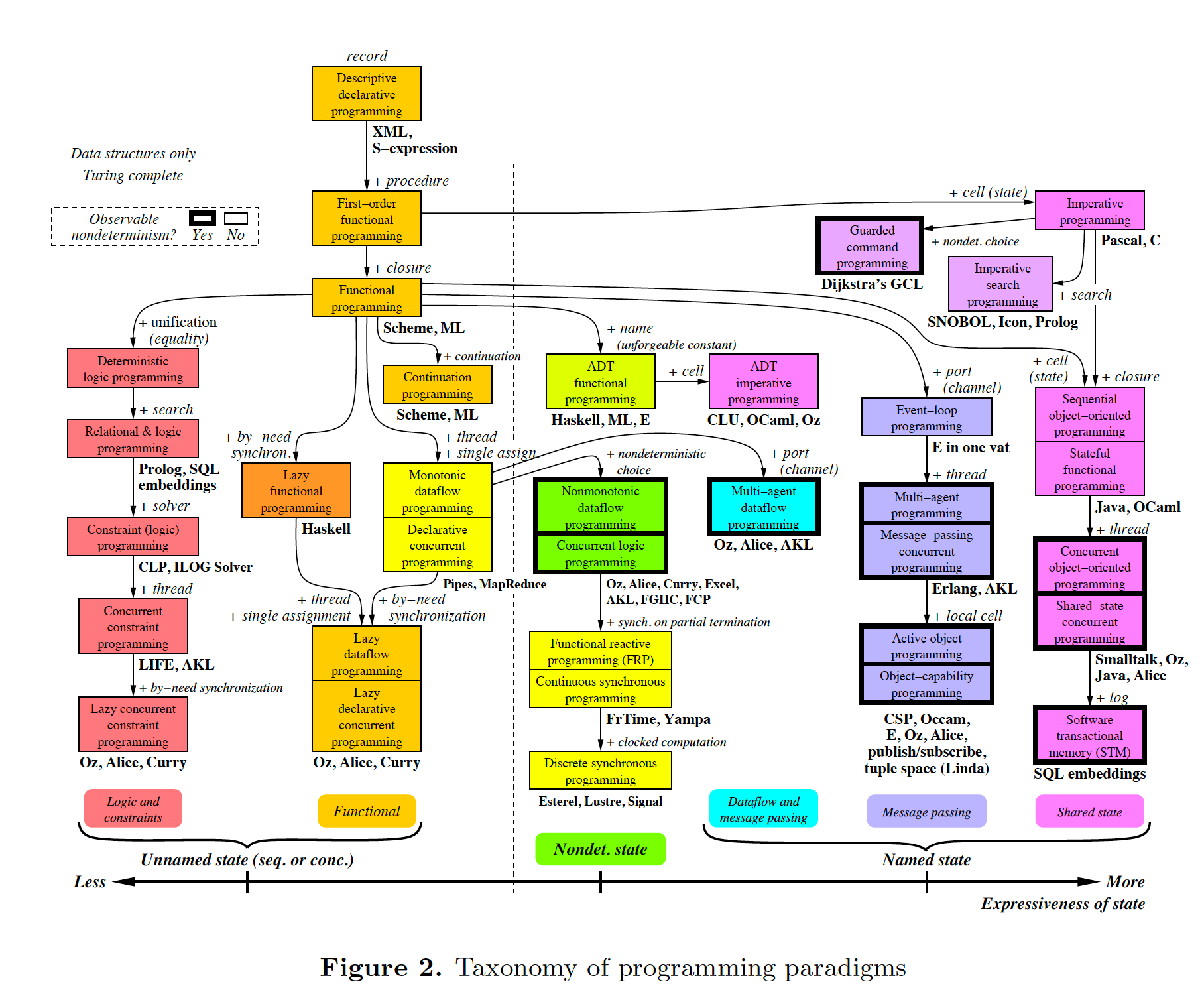
5. [Andreas Zwinkau](https://beza1e1.tuxen.de/index.html), *Faster Than C*, 24. februāris 2013. gads [skatīts 14. decembrī 2019. gadā], Pieejams: <https://beza1e1.tuxen.de/articles/faster_than_C.html>

6. [Andrew Meredith](https://dev.to/kendru)*, Can Rust replace C/C++ as system language?,* 12. septembris 2019. gads [skatīts 14. decembrī 2019. gadā], Pieejams: <https://dev.to/kendru/can-rust-replace-c-c-as-a-systems-language-7c7>

1. Professor Marsette (Marty) Vona, *Computer Graphics Suggested Platforms*, 21. decembris 2019. gads [skatīts 21. decembrī 2019. gadā] Pieejams: http://www.ccs.neu.edu/home/fell/CS5310/platforms.html
2. Alexander Medvednikov, *Comparison of V and Other Languages,* februāris 2019. gads [skatīts 11. decembrī 2019. gadā], Pieejams: <https://vlang.io/compare>
3. Steve Klabnik un Carol Nichols, *The RUST programming language,* 2018. gads, 617. lpp. Pieejams: <https://lise-henry.github.io/books/trpl2.pdf>
4. Miek Gieben, *Learning Go,* 2012. gads 112. lpp. Pieejams: <https://miek.nl/files/go/Learning-Go-latest.pdf>
5. Alexander Medvednikov, *The V Programming Language*, februāris 2019. gads [skatīts 9. februārī 2020. gadā], Pieejams: <https://vlang.io/>
6. Mark Segal, Kurt Akeley, Chris Frazier, Jon Leech, Pat Brown, *OpenGL 4.6 Core Profile*, Michigan, 22. oktobris 2019. gads, 850 lpp Pieejams: https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf

PIELIKUMI

1. pielikums  
   Paradigmas taksonomijas attēls[[19]](#footnote-19)



Es, Aleksandrs Koroļko, bakalaura studiju programmas „Informācijas tehnoloģija” III RDBI01 grupas students(-e), ar savu parakstu apstiprinu, ka esmu izstrādājis(-usi) doto bakalaura darbu, kas iesniegts Rīgas Tehniskajā universitātē inženierzinātņu bakalaura grāda datorvadībā un datorzinātnē iegūšanai.

Bakalaura darbs ir izpildīts pilnīgi patstāvīgi un satur visas nepieciešamās atsauces uz darbā izmantotajiem materiāliem.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(studenta paraksts)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Bakalaura darbs izstrādāts Vadības informācijas tehnoloģijas katedrā

Darba autors: Aleksandrs Koroļko \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts)

201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Zinātniskais vadītājs: Artūrs Braučs \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (paraksts)

201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Bakalaura darbs pielaists aizstāvēšanai:

Vadības informācijas tehnoloģijas katedras vadītājs: Jānis Grabis \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts)

201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Bakalaura darbs aizstāvēts Informācijas tehnoloģijas institūta bakalaura darbu aizstāvēšanas komisijas 201\_. gada \_\_\_.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ sēdē, protokola Nr. \_\_\_\_\_\_\_\_ un novērtēts ar atzīmi \_\_\_\_\_\_\_\_\_.

Informācijas tehnoloģijas institūta bakalaura darbu aizstāvēšanas komisijas sekretāre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /doc.V.Minkēviča/

1. Līdzīgas produkcijas apskatīšana un salīdzināšana tīmekļa veidne / Ir pieejams [www.slant.co](http://www.slant.co) [↑](#footnote-ref-1)
2. Tīmekļa raksts “What exactly is a programming paradigm?” / autors M.V.Thanoshan / datums [12. novembris 2019. gads] / Ir pieejams <https://www.freecodecamp.org/news/what-exactly-is-a-programming-paradigm/> [↑](#footnote-ref-2)
3. Tīmekļa raksts “” / autors Bhumika Rani/ Ir pieejams <https://www.geeksforgeeks.org/introduction-of-programming-paradigms/> [↑](#footnote-ref-3)
4. Tīmekļa raksts “Programming paradigms” / Ir pieejams <https://cs.lmu.edu/~ray/notes/paradigms/> [↑](#footnote-ref-4)
5. Tīmekļa IBM raksts par Rust / autors M. Tim Jones / datums [8. marts 2018. gadā] / ir pieejams <https://developer.ibm.com/technologies/web-development/articles/os-developers-know-rust/> [↑](#footnote-ref-5)
6. Attēla ir pieejama <https://developer.ibm.com/developer/articles/os-developers-know-rust/images/Figure01.png> [↑](#footnote-ref-6)
7. Tīmekļa raksts “Rust for JavaScript Developers” / autors David Morcillo / datums [15. oktobris 2019. gads] / Ir pieejams <https://www.codegram.com/blog/rust-for-js-developers/> [↑](#footnote-ref-7)
8. Tīmekļa raksts “Getting Started With Rust, the friendly System Programming Language” / autors Dhruvil Trivedi / datums [21. jūlijs 2018. gads] / Ir pieejams <https://opensourceforu.com/2018/07/getting-started-with-rust-the-friendly-systems-programming-language/> [↑](#footnote-ref-8)
9. Attēla ir pieejama <https://i2.wp.com/opensourceforu.com/wp-content/uploads/2018/07/Figure-2-Graphical-comparison-of-Rust.jpg?ssl=1> [↑](#footnote-ref-9)
10. Rust FAQ par lēnu kompilāciju ir pieejama [https://prev.rust-lang.org/en-US/faq.html#why-is-rustc-slow](https://prev.rust-lang.org/en-US/faq.html%23why-is-rustc-slow) [↑](#footnote-ref-10)
11. Tīmekļa raksts “Rust – memory safety without garbage collector” / autors Yan Cui / datums [5. aprīlis 2017. gads] / Ir pieejams - https://medium.com/theburningmonk-com/rust-memory-safety-without-garbage-collector-d6a25b23c036 [↑](#footnote-ref-11)
12. Tīmekļa atskaite par GO Lang programmēšanas valodu / Ir pieejams - <https://kuree.gitbooks.io/the-go-programming-language-report/content/16/text.html> [↑](#footnote-ref-12)
13. Tīmekļa raksts “Why should you learn GO?” / autors Kavel Patel / datums [8. janvārī 2017. gads] / Ir pieejams - <https://medium.com/@kevalpatel2106/why-should-you-learn-go-f607681fad65> [↑](#footnote-ref-13)
14. Apmācības GO Lang veidnē / Ir pieejams - <https://www.golang-book.com/books/intro/10> [↑](#footnote-ref-14)
15. Oficiāla FAQ atbilde uz GO Lang mērķi - [https://golang.org/doc/faq#What\_is\_the\_purpose\_of\_the\_project](https://golang.org/doc/faq%23What_is_the_purpose_of_the_project) [↑](#footnote-ref-15)
16. Attēla ir pieejama šeit - [https://miro.medium.com/max/994/1\*xbsHBQJReC5l\_VO4XgNSIQ.png](https://miro.medium.com/max/994/1*xbsHBQJReC5l_VO4XgNSIQ.png) [↑](#footnote-ref-16)
17. Pilna C Lang nepieciešama vieta ar kompilatoru - https://lists.llvm.org/pipermail/llvm-dev/2019-April/132028.html [↑](#footnote-ref-17)
18. Pilna Swift nepieciešama vieta ar kompilatoru - https://github.com/apple/swift#getting-started [↑](#footnote-ref-18)
19. Ir pieejams <https://medium.com/@jingchenjc2019/a-brief-survey-of-programming-paradigms-207543a84e2b> [↑](#footnote-ref-19)